

Liviu KREINDLER
Raducu GIUCLEA



BAZELE MICROPROCESOARELOR

MATRIX
ROM

BUCUREȘTI 1997

CUVANT INAINTE

Prezenta lucrare trateaza principalele aspecte legate de sistemele cu microprocesoare, privite din punctul de vedere al inginerului de aplicatie, cu profil electrotehnic. Lucrarea se adreseaza viitorilor utilizatori ai unor sisteme cu microprocesoare, care vor fi pusi in situatia de a configura asemenea sisteme, de a implementa aplicatii dedicate, prin proiectarea interfetelor specifice si elaborarea programelor aferente.

Structura prezentata reprezinta in esenta cursul de “Sisteme cu microprocesoare” predat de catre autori studentilor de la anul IV ai facultatii de Electrotehnica a Universitatii “Politehnica” din Bucuresti.

Scopul principal al cartii este de a oferi informatiile si a forma conceptele fundamentale legate de problematica utilizarii sistemelor microprocesor. Diversitatea tipurilor, caracteristicilor de lucru si performantelor diverselor familii de microprocesoare, ritmul extrem de ridicat in care apar noi componente, tehnologii, concepte, impun utilizarea unei anumite strategii in abordarea acestui subiect. S-a urmarit definirea notiunilor general valabile pentru toate aceste sisteme, pe baza intelegerii carora inginerul de aplicatie va putea aprofunda, pentru configuratii concrete, elementele specifice asociate acestor configuratii.

In capitolul 1, “Introducere”, se face o trecere in revista a istoricului microprocesoarelor, a evolutiei acestora in cele peste doua decenii scurse de la aparitie, a tehnologiilor, tipurilor si performantelor acestora. Se definesc sistemele numerice operand in timp real, si tipul reprezentativ de aplicatie asociat acestora.

Capitolul 2 “Notiuni introductive”, prezinta principalele notiuni legate de aritmetica numerelor intregi, si a elementelor de logica circuitelor numerice. Aceste elemente sunt fundamentale pentru intelegerea si studiul sistemelor cu microprocesoare.

Capitolul 3, “Microprocesoare - Concepte generale”, reprezinta o ampla analiza a conceptelor legate de microprocesoare si de sistemele cu microprocesoare. Se analizeaza atat microprocesorul, din punct de vedere functional, si al structurii hardware si software de baza, cat si structura si principiile de configurare a memoriei si interfetelor de intrare-iesire ale unui sistem microprocesor. Capitolul se incheie cu un exemplu de sistem microprocesor simplificat, ce implementeaza o aplicatie simpla, dar completa, continand atat specificatiile de proiectare hardware cat si software necesare.

Capitolul 4 abordează prezentarea circuitului INTEL 8086, tip reprezentativ de microprocesor, atât datorită structurii și caracteristicilor sale hardware și a setului de instrucțiuni, cât și a utilizării pe scară largă în construcția microcalculatoarelor de tip PC.

Capitolul 5, “Implementarea interfețelor de intrare-iesire”, prezintă aspecte legate de implementarea porturilor de uz general de tip numeric sau analogic, cât și a principalelor tipuri de interfețe de intrare-iesire specializate, de tipul porturilor seriale, ceasurilor de timp real, sau a controlerelor de întrerupere. Se fac exemplificări ce utilizează componente compatibile cu microprocesorul INTEL 8086, schemele și programele prezentate fiind implementabile pe microcalculatoare de tip PC-AT.

Capitolul 6 “Aplicații ale sistemelor microprocesor”, încununează acest studiu cu prezentarea unui exemplu complet de proiectare a unui sistem de control discret implementat cu microprocesor de tip 8086. Pornind de la enunțul temei aplicației, după o analiză a structurii, alegere a componentelor și definirii parametrilor acestora, se proiectează structura sistemului de control - atât constructivă - hardware, cât și programul aferent - software. Furnizarea tuturor detaliilor de proiectare oferă astfel cititorului posibilitatea de a cunoaște în mod concret principiile pe baza cărora trebuie acționat pentru abordarea unei asemenea problematice.

Autorii speră ca această lucrare să reprezinte un instrument util pentru înțelegerea problematicei de bază referitoare la utilizarea, configurarea și programarea sistemelor cu microprocesoare utilizate în aplicațiile industriale. Sperăm că gradul de generalitate la care s-a încercat să se pastreze prezentarea va fi util prin oferirea de informații valabile pentru cea mai mare parte a sistemelor cu microprocesoare existente, și astfel, va oferi un punct de pornire celor interesați în înțelegerea principiilor și în utilizarea cu performanțe maxime a unor asemenea sisteme.

CUPRINS

1. INTRODUCERE.....	1-1
1.1. MICROPROCESORUL - REPERE GENERALE	1-1
1.2. OBIECTUL CARTII	1-5
2. NOTIUNI INTRODUCTIVE	2-1
2.1 ARITMETICA NUMERELOR INTREGI.....	2-1
2.1.1 Aritmetica numerelor intregi, pozitive (fara semn).....	2-3
2.1.2 Aritmetica numerelor intregi, cu semn.....	2-4
2.1.3 Aritmetica numerelor zecimale codificate binar (BCD).....	2-6
2.1.4 Aritmetica numerelor fractionare	2-6
2.2 ELEMENTE DE LOGICA CIRCUITELOR NUMERICE	2-6
2.2.1 Logica booleana	2-7
2.2.2 Porti logice.....	2-8
2.2.3 Inversoare.....	2-9
2.2.4 Reprezentari logice alternative	2-9
2.2.5 Poarta tri-state	2-10
2.2.6 Bistabilul de tip D.....	2-10
2.2.7 Registre.	2-12
2.2.8 Multiplexoare	2-14
2.2.9 Demultiplexoare si decodificatoare	2-15
2.2.10 Circuite logice programabile (PLD)	2-16
3. MICROPROCESOARE - CONCEPTE GENERALE.....	3-1
3.1 MICROPROCESORUL --- PREZENTARE FUNCTIONALA	3-2
3.1.1 Functiile de baza ale microprocesorului	3-3
3.1.2 Aspectul temporal - secvential	3-6
3.1.3 Reprezentarea informatiei	3-6
3.1.4 Memorarea informatiei	3-7
3.1.5 Conceptul de program.....	3-9
3.2 CONCEPTUL DE MAGISTRALA.....	3-9
3.2.1 Magistrala de date	3-11
3.2.2 Magistrala de adrese	3-11
3.2.3 Magistrala de control	3-12
3.3 MICROPROCESORUL - PREZENTARE <i>HARDWARE</i>	3-13
3.3.1 Functia de control.....	3-13
3.3.2 Functia de memorare	3-22
3.3.3 Functia aritmetico-logica	3-25
3.3.4 Functia de intrare/iesire.....	3-28
3.3.5 Microprogramarea	3-29
3.4 MEMORIA INTERNA A SISTEMULUI MICROCALCULATOR.....	3-31

3.4.1	Memorii RAM	3-32
3.4.2	Memorii ROM	3-40
3.4.3	Organizarea tipica a memoriei sistemului microprocesor	3-43
3.5	SOFTWARE-UL SISTEMULUI MICROPROCESOR-ASPECTE GENERALE	3-44
3.5.1	Consideratii de baza	3-45
3.5.2	Moduri de adresare	3-46
3.5.3	Setul de instructiuni al microprocesorului	3-54
3.5.4	Utilizarea stivei programului	3-67
3.5.5	Limbajul de asamblare	3-70
3.5.6	Metodologia elaborarii programelor	3-73
3.6	ELEMENTE DE INTRARE/IESIRE ALE SISTEMELOR MICROPROCESOR	3-75
3.6.1	Operatii I/E efectuate sub controlul programului	3-77
3.6.2	Operatii de intrare/iesire efectuate prin intreruperi	3-82
3.6.3	Operatii de I/E efectuate prin <i>acces direct la memorie</i>	3-87
3.7	SISTEM MICROPROCESOR --- EXEMPLU	3-89
3.7.1	Structura hardware a sistemului	3-89
3.7.2	Structura software a sistemului	3-93
3.7.3	Concluzii	3-97
4.	MICROPROCESORUL INTEL 8086	4-1
4.1	ARHITECTURA PROCESORULUI	4-2
4.1.1	EU — Unitatea de executie	4-3
4.1.2	BIU — Unitatea de interfata cu magistrala	4-4
4.1.3	Registrele de uz general	4-4
4.1.4	Registrele de segment	4-6
4.1.5	Registru pointerului de instructiuni	4-6
4.1.6	Indicatorii de conditie	4-6
4.2	ORGANIZAREA MEMORIEI	4-8
4.2.1	Segmentarea memoriei	4-9
4.2.2	Generarea adresei fizice	4-10
4.2.3	Stiva	4-12
4.3	ORGANIZAREA SPATIULUI DE INTRARE/IESIRE	4-15
4.3.1	Spatiu de intrare/iesire propriu-zis	4-15
4.3.2	Porturile de I/E plasate in spatiul de memorie	4-15
4.4	SPECIFICATII HARDWARE	4-15
4.4.1	Definirea pinilor	4-15
4.4.2	Functionarea magistralei multiplexate	4-17
4.4.3	Circuitul de ceas	4-23
4.4.4	Modul minim/maxim. Controlerul de magistrala 8288	4-24
4.4.5	Selectia memoriei	4-29
4.4.6	Selectia si interfatarea porturilor de intrare/iesire	4-32
4.5	Sistemul de intreruperi	4-34
4.5.1	Sursele de generare a intreruperilor	4-34
4.5.2	Vectorii de intrerupere	4-35
4.5.3	Intreruperile externe	4-37
4.5.4	Intreruperile interne	4-40
4.5.5	Subrutina de tratare a intreruperii	4-41

5.	IMPLEMENTAREA INTERFETELOR DE INTRARE / IESIRE	5-1
5.1	IMPLEMENTAREA PORTURILOR DE INTRARE-IESIRE NUMERICE	5-1
5.1.1	Consideratii generale privind porturile de I/E numerice	5-1
5.1.2	Exemplu de schema de porturi de I/E numerice	5-2
5.1.3	Exemple de programarea porturilor de I/E numerice	5-6
5.2	IMPLEMENTAREA PORTURILOR DE INTRARE-IESIRE ANALOGICE	5-7
5.2.1	Implementarea interfetei convertorului analog-digital.....	5-7
5.2.2	Implementarea interfetei convertorului digital-analog.....	5-8
5.3	PORTURI SPECIALIZATE – INTERFETE SERIALE	5-10
5.3.1	Interfete seriale - notiuni generale	5-10
5.3.2	Interfata seriala INTEL 8251	5-13
5.3.3	Porturile de comunicatie seriala la calculatoarele de tip PC-AT.....	5-15
5.3.4	Exemplu de programare a portului serial COM2	5-17
5.4	PORTURI SPECIALIZATE – CEASURI DE TIMP REAL	5-18
5.4.1	Ceasuri programabile – notiuni generale.....	5-18
5.4.2	Ceasul programabil INTEL 8253	5-19
5.4.3	Ceasurile programabile la calculatoarele de tip PC-AT	5-22
5.4.4	Exemplu de utilizare a ceasurilor programabile la calculatoarele de tip IBM-PC	5-24
5.5	PORTURI SPECIALIZATE – CONTROLERE DE INTRERUPERI	5-28
5.5.1	Programarea controlerului de intreruperi 8259A	5-28
5.5.2	Tratarea intreruperilor externe mascabile in microcalculatoarele PC-AT.....	5-30
5.5.3	Exemple de programe de tratarea intreruperilor	5-31
6	APLICATII ALE SISTEMELOR MICROPROCESOR	6-1
6.1	PERFORMANTELE IMPUSE SISTEMULUI DE CONTROL NUMERIC. ALEGEREA CONFIGURATIEI SISTEMULUI.	6-3
6.1.1	Sarcina	6-4
6.1.2	Ciclul de miscare	6-5
6.1.3	Performantele de regim dinamic	6-5
6.1.4	Precizia sistemului reglat.....	6-6
6.1.5	Viteza de lucru a sistemului de control	6-6
6.1.6	Precizia traductoarelor.....	6-6
6.1.7	Precizia de calcul.....	6-7
6.1.8	Alegerea componentelor de forta	6-7
6.1.9	Alegerea traductoarelor	6-9
6.1.10	Alegerea tipului de schema de reglare	6-11
6.1.11	Alegerea microprocesorului.....	6-11
6.2	PROIECTAREA SCHEMEI DE CONTROL. ASPECTE DE PRINCIPIU.	6-12
6.2.1	Obtinerea modelului sistemului	6-13
6.2.2	Discretizarea modelului sistemului	6-14
6.2.3	Proiectarea controlerului discret.....	6-15
6.3	PROIECTAREA STRUCTURII HARDWARE A SISTEMULUI	6-17
6.3.1	Memoria PROM.....	6-20
6.3.2	Memoria RAM.....	6-22
6.3.3	Porturile de intrare-iesire	6-24
6.4	PROIECTAREA STRUCTURII SOFTWARE A SISTEMULUI.....	6-37

6.4.1	Specificatiile de programare ale resurselor sistemului	6-37
6.4.2	Organizarea programului.....	6-41
6.4.3	Prezentarea programului implementat	6-41
6.4.4	Concluzii	6-51

1. INTRODUCERE

1.1. MICROPROCESORUL - REPERE GENERALE

Printre progresele tehnologiei electronice, înregistrate în ultimele decenii, apariția microprocesoarelor a reprezentat un moment de cotitură, atât în ceea ce privește plaja practic nelimitată a domeniilor de aplicatie a acestor componente revolutionare din punct de vedere tehnic și tehnologic, cât și în ceea ce privește concepția și realizarea (tehnicile de analiză, proiectare, sintetizare și testare) a sistemelor cu microprocesor. Un nou mod de abordare a controlului proceselor industriale, cu ajutorul sistemelor numerice, devenea posibil cu acest prilej.

Începând modest, cu microprocesoare de 4 biți, de viteză relativ redusă, cu costuri ridicate și dispunând de relativ puține elemente de dezvoltare a aplicațiilor, piața microprocesoarelor a înregistrat progrese de-a dreptul înimaginabile în urma cu câțiva ani, evoluând într-o progresie exponențială.

Obținerea "calculatoarelor pe un cip", cu performanțe de-a dreptul spectaculoase, s-a făcut extrem de rapid, iar estimările pentru următorii ani prevăd o expansiune a sistemelor microprocesor, ce vor depăși, în curând, ca număr de sisteme implementate în diverse aplicații, populația globului pământesc.

Și din punctul de vedere al caracteristicilor putere de calcul / cost / nivel de sofisticare, microprocesoarele și microcalculatoarele (calculatoarele care utilizează ca unitate centrală un microprocesor), se situează pe poziții deosebit de avantajate, comparativ cu celelalte tipuri "clasice", de calculatoare. Figura 1.1. prezintă sintetic unele dintre aceste caracteristici pentru diversele tipuri de echipamente de calcul aflate în uz. Se remarcă faptul că microcalculatoarele și microprocesoarele reprezintă elemente deosebit de atractive din punctele de vedere al raportului performanță/preț, și nivel de sofisticare/preț, în comparație cu celelalte tipuri de asemenea echipamente.

Termenul de "microprocesor" a fost introdus în 1972 de către firma americană INTEL, realizatoarea primului microprocesor de 4 biți, în anul 1971. Curând, microprocesoare de 8, 16 și 32 de biți au fost produse (în 1972, 1974 și 1981, respectiv). Numărul de componente pe cip a crescut de peste 500 de ori, frecvența de lucru a circuitelor de peste 100 de ori, în esență capacitatea globală a acestor sisteme s-a majorat cu peste trei ordine de mărime.

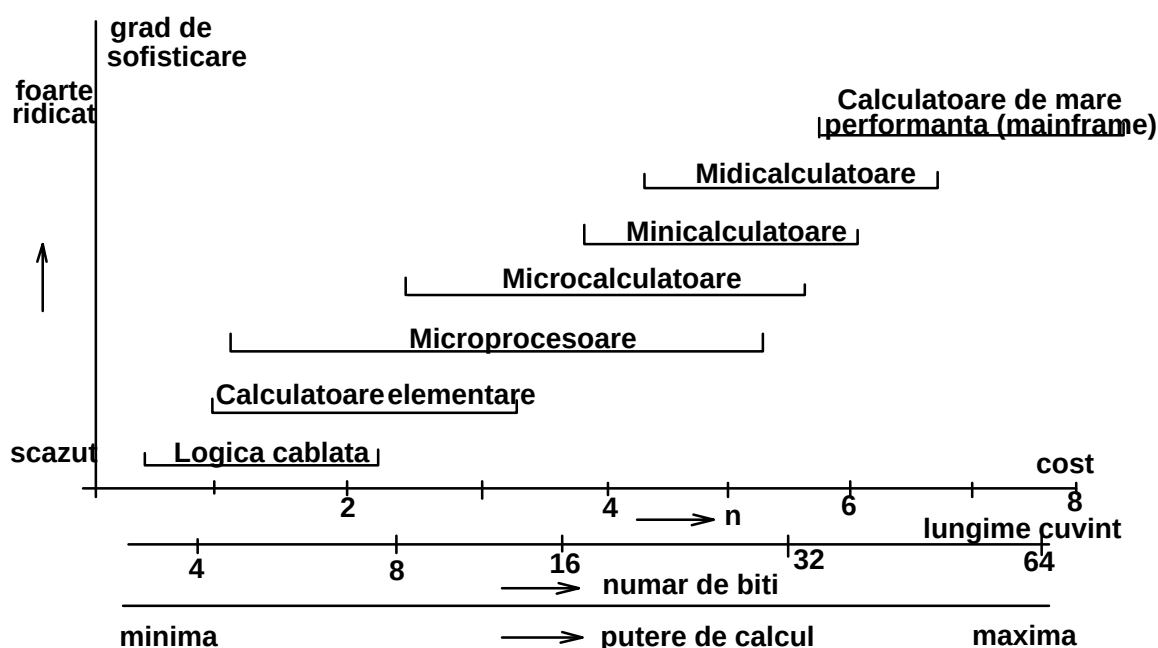


Fig.1.1. Compararea costului, puterii de calcul si capacitatii diverselor calculatoare numerice

In primul deceniu de existenta, microprocesoarele au evoluat de la structuri relativ elementare, cu performante limitate, la situatia de a se impune treptat in fruntea sistemelor de calcul de larga raspandire, atat din punct de vedere al puterii de calcul, cat si al structurii arhitecturale, deosebit de sofisticate. Primele tipuri de microprocesoare indeplineau doar functiile de baza ale sistemului, necesitand suplimentar o gama intreaga de circuite suplimentare pentru generarea semnalelor de temporizare, pentru memorarea datelor si a programelor, pentru interfatarea cu perifericele sistemului si cu procesul controlat, etc. Utilizarea microprocesoarelor a fost orientata initial cu predilectie ca *microcontrolere* in structuri dedicate, pentru aplicatii de control al proceselor. Setul de instructiuni al acestora era relativ limitat, iar programarea se facea direct in cod masina. Dezvoltarea ulterioara a arhitecturilor evolute, aparitia limbajelor de programare de nivel inalt, deosebit de performante, cresterea puterii de calcul si a flexibilitatii, aparitia si expansiunea sistemelor de dezvoltare, au largit considerabil aria de aplicatii ale microprocesoarelor.

Din punctul de vedere al *tehnologiei de fabricatie*, s-a pornit initial de la tehnologia pMOS. Practic, toate tehnologiile disponibile de fabricare a circuitelor semiconductoare au fost testate de-a lungul anilor (CMOS, NMOS, bipolare, I²L). Sunt preferate circuitele realizate in tehnologii MOS, datorita densitatii mai mari de tranzistoare pe cip, ce se pot obtine in aceste tehnologii. Dintre acestea, tehnologia NMOS a fost cea mai utilizata, atit datorita densitatii mari cat si a vitezei ridicate de comutatie. (Firma INTEL a denumit aceasta tehnologie HMOS - H $\leftrightarrow$ *high performance* = mare performanta). Astfel, circuitul INTEL 8086, produs in 1978, continea 29000 de tranzistoare; nu mult mai tarziu, in 1982, era produs circuitul INTEL 80286, cu un numar de 130000 de tranzistoare pe cip. Aparut in 1988, microprocesorul 80486 a ajuns la peste 2000000 tranzistoare pe cip. Progrese tot mai mari s-au facut in tehnologia

CMOS, care prezinta avantajul de a avea viteze superioare, consum redus de putere, si o mai mare imunitate la zgomot.

Unul dintre cele mai importante criterii, pentru evaluarea si selectia microprocesoarelor, este marimea cuvantului de date al acestora. Cuvinte de date mai mari implica evident o putere de calcul si adresabilitate sporite. Actualmente, microprocesoarele de 16 si 32 biti domina piata de microcalculatoare, desi o gama relativ importanta de aplicatii, in domeniul controlului proceselor industriale, este implementata cu microprocesoare de putere mai mica (uzual, microprocesoare de 8 biti), suficiente ca putere de calcul si capacitati de adresare, si competitive ca pret de cost.

Progrese importante s-au facut si in domeniul "microprogramarii". Procesoarele microprogramate, (spre deosebire de microprocesoarele "clasice", cu structura fixa si set de instructiuni impus de fabricant), permit (desi la o viteza relativ mai redusa), obtinerea unor structuri mult mai flexibile, in ceea ce priveste realizarea si modificarea eventuala a setului de instructiuni, de catre utilizator, adaptate specific aplicatiei abordate. Aceste instructiuni (macro-instructiuni), sunt definite pe baza unui set restrans de microinstructiuni, de catre realizatorul sistemului. Executia programelor se face astfel in doua etape succesive (doua nivele de control). Aceasta reduce intru-catva viteza globala a sistemului.

Progresele inregistrate din punct de vedere tehnologic au facut ca, practic, *densitatea* circuitelor integrate de tip VLSI¹ ce includ microprocesoarele, sa se dubleze la fiecare doi ani. Avantajul imediat al acestui fapt consta in posibilitatea de a se implementa seturi de instructiuni mult imbunatatite si extinse. Exista astfel microprocesoare in care o singura instructiune controleaza transferul unor blocuri de date din memorie, sau opereaza cu mai multe registre simultan. De multe ori, unele instructiuni sunt apropiate de instructiuni in limbaje de nivel inalt, usurand compilarea deosebit de eficienta a programelor, si implementand direct o gama de functii uzual realizate prin programare (*software*). Cresterea performantelor si a complexitatii acestor circuite s-a facut bineinteles cu pretul unui efort de proiectare sensibil crescut. Astfel, acesta a crescut de la 1 om-an, la peste 100 om-an, pentru circuitele cele mai avansate.

In decursul anilor, a crescut sensibil si capacitatea de adresare a microprocesoarelor. Numarul de moduri auxiliare de adresare a crescut deasemenea. Multe firme ofera circuite specializate pentru controlul si coordonarea functionarii memoriei sistemului. Totodata, firme ca INTEL includ in circuitul microprocesorului si functiile de protectie a accesului si control al memoriei. Utilizarea *memoriei virtuale* permite adresarea unor spatii de ordinul gigaoctetilor. Este posibila astfel executia programelor foarte mari, executia "simultana" a mai multor programe (*multitasking*), lucrul "simultan" al mai multor utilizatori (*multiuser*). Asemenea sisteme utilizeaza suporturi de memorie exterioara (disc), pentru memorarea temporara a programului si a datelor. Un concept deasemenea tot mai utilizat este asa-numita *memorie cache*, relativ mai redusa volumic, dar de viteze foarte mari, in care se inmagazineaza programul sau datele, extrase din memoria sistemului, sau de pe disc, marind sensibil, per global, viteza de lucru a sistemului, si deci performantele acestuia.

¹ VLSI - Very Large Scale Integrated = Integrare pe scara foarte larga

Microprocesoarele diferă sensibil în ceea ce privește *capacitățile de memorare și manipulare* a diverselor tipuri de date. Dacă reprezentarea și manipularea informațiilor sub forma octetilor și a cuvintelor de date este uzuală la toate tipurile de microprocesoare, alte tipuri de operații, ca de exemplu cu informații reprezentate pe biti, BCD¹, reale (virgula mobilă), texte (șiruri de caractere), etc., nu sunt întotdeauna direct acceptate. De exemplu, date manipulate ca biti individuali nu sunt acceptate de către microprocesorul 8086, deși sunt foarte utilizate în aplicațiile de proces. Cu cât acceptă o gamă mai extinsă de tipuri de date, microprocesorul devine mai potrivit unei game tot mai largi de posibile aplicații. Din neferică această implică o structură complexă a sistemului, și utilizarea unui număr tot mai mare de componente. Uneori, tipuri de date suplimentare se pot folosi doar prin utilizarea unor circuite auxiliare. De exemplu, circuitul INTEL 8087, coprocesor matematic compatibil cu microprocesorul 8086, permite implementarea calculelor în virgula mobilă în sistemele respective. Diversitatea tipurilor de microprocesoare și a coprocesoarelor aferente, produse de diferite firme, a impus standardizarea operațiilor aritmetice în virgula mobilă (standardul IEEE P754), acceptată de către microprocesoarele sau coprocesoarele aritmetice aferente.

Revoluția microprocesoarelor este generată de tendința de a se implementa toate componentele unui calculator cu ajutorul unui număr relativ redus de cipuri. Orice calculator, indiferent de mărime, încorporează trei componente de bază: **unitatea centrală de prelucrare (CPU²)**, continuând elementele de calcule aritmetice, logice și funcțiile de control; **memoria (RAM** - pentru citire/scriere, și **ROM** - doar citire), și **interfețele de intrare/ieșire** pentru controlul periferiei calculatorului. Odată cu progresele tehnologice înregistrate, tot mai multe componente pot fi integrate pe același cip, și astfel tot mai multe funcții auxiliare se pot implementa pe cipul microprocesorului însuși. Se obțin astfel calculatoare construite cu foarte puține cipuri. Un **microcalculator** combină un microprocesor cu memorie și capacități de intrare/ieșire pe unul sau mai multe cipuri. **Microcalculatoarele pe un singur cip** constituie un subset important al microprocesoarelor, în care toate funcțiile, inclusiv memoria, sunt implementate pe același cip. Evident, apariția acestor tipuri de microcalculatoare este întotdeauna relativ întârziată față de apariția microprocesoarelor cu o putere de calcul similară (de exemplu, primul microcalculator de 8 biti pe un cip, INTEL 8048, a fost produs în anul 1976, la patru ani după producerea primului microprocesor de 8 biti, INTEL 8008).

Deoarece aceste circuite se produc în cantități deosebit de mari, este necesară o proiectare foarte atentă, optimizându-se arhitectura propusă, memoria sistemului, setul de instrucțiuni, tehnicile de adresare a memoriei, și alte restricții de proiectare, legate de parametrii de performanță, execuție, etc. Peste 4/5 din tranzistoarele cipului tind să fie utilizate pentru funcții de memorie. Deși au performanțe foarte bune, aceste circuite sunt încă departe de performanțele de calcul ale supercalculatoarelor de tipul CRAY-1.

La cealaltă extremitate a spectrului se află aplicațiile ce necesită o putere de calcul foarte mare, sau precizie sporită, ce nu se pot atinge cu microprocesoarele "clasice". Utilizarea procesoarelor *bit-slice* permite legarea mai multor cipuri modulare identice în paralel, până la

¹ BCD - Binary Coded Decimal = zecimal codat binar

² CPU - Central Processing Unit = unitate centrală de prelucrare

obtinerea structurii cu o precizie corespunzatoare. Dupa cum s-a precizat, posibilitatea de a dezvolta si un macro-set de instructiuni, specific unei aplicatii date, confera un avantaj suplimentar la utilizarea acestor structuri.

O tot mai mare amploare o au in ultima vreme procesoarele specializate, de tipul procesoarelor **DSP**¹, cu capacitati de calcul deosebite (implementare specifica de filtre numerice, transformate *Fourier* - *FFT*, corelatii, convolutii, etc.), in ceea ce priveste prelucrarea numerica a semnalelor. Uzual, aceste microprocesoare sunt dotate cu elementele de interfatare necesare, inclusiv convertoare *A/D* si *D/A* pentru conectarea la procese analogice.

O evolutie interesanta au avut-o si transputerele, procesoare cu structuri pretabile la realizarea de calculatoare cu procesare paralela, de performante deosebite de ridicate, mai ales in ceea ce privesc performantele de calcul obtinute.

In fine, trebuiesc mentionate circuitele integrate de tip *VLSI*, proiectate si realizate direct pentru o aplicatie data (asa numitele circuite de tip **ASIC**²). S-a ajuns la solutii economice convenabile de a se proiecta, pe un sistem microcalculator de putere sporita (asa numitele **workstations**³), utilizand pachete puternice de programe de proiectare (**CAD**⁴), structura circuitului dorit. Se porneste de la caracteristicile functionale finale dorite ale circuitului. Proiectantul trebuie sa cunoasca cerintele produsului final, si sa il sintetizeze utilizand ca elemente componente "blocuri" functionale predefinite, puse la dispozitia sa de catre programul de proiectare. Dupa definirea structurii functionale a sistemului, se simuleaza functionarea si se modifica (daca este cazul) structura circuitului, pana la obtinerea comportarii si performantelor dorite. Apoi, se produc efectiv circuitele respective, si se testeaza "pe viu" functionarea acestora. Acest mod de proiectare ofera avantajul deosebit de a se muta conceperea circuitului, la virtualul utilizator, care poate produce cel mai adecvat procesor pentru aplicatia data. Bineinteles, economic vorbind, acest mod de producere a circuitelor integrate specializate (inclusiv procesoare), este avantajos doar pentru cantitati relativ mari de circuite de un tip dat. Durata proiectarii si realizarii circuitelor in acest mod scade sensibil.

Concluzionand, putem remarca extrema diversitate de aspecte ridicate de caracterizarea circuitelor microprocesor existente. Paleta deosebit de larga de performante, caracteristici, structuri si nu in ultimul rand, de preturi de cost, poate pune probleme serioase in fata utilizatorului, a proiectantului de sistem, care trebuie sa decida, alegand solutia optima, conform unui criteriu complex de evaluare a sistemului.

1.2. OBIECTUL CARTII

¹ DSP - Digital Signal Processing = prelucrarea numerica a semnalelor

² ASIC - Application Specific Integrated Circuit = circuit integrat specific unei aplicatii date

³ workstations = statii de lucru

⁴ CAD - Computer Aided Design = proiectare asistata de calculator

Pe baza considerentelor prezentate în paragraful anterior, putem totuși concluda că "filozofia" generală a realizării sistemelor microcalculator, componentele tipice, funcțiile de bază, sunt comune tuturor tipurilor de microprocesoare, indiferent de firma producătoare și de generația de circuite careia îi aparține microprocesorul.

Plecând de la aceste premize, și totodată luând în considerare faptul că, pentru acest volum, punctul de pornire, cât și cel de sosire în studiul microprocesoarelor, este cel al aplicațiilor industriale, și cu precădere cel al aplicațiilor în echipamentele electrice (acționari electrice, sisteme de achiziții de date, standuri de măsură, diagnoză, control, etc.), prezentul volum se dorește o încercare de a pune bazele conceptuale ale proiectării sistemelor cu microprocesoare, de a fundamenta principalele aspecte și probleme legate de structura constructivă (hardware) și programele implementate (software) în aceste sisteme.

Într-un mod oarecum general, acoperind practic o gamă extrem de largă de cazuri, figura 1.2 prezintă structura de principiu a unui asemenea sistem, controlat cu ajutorul unui microcalculator.

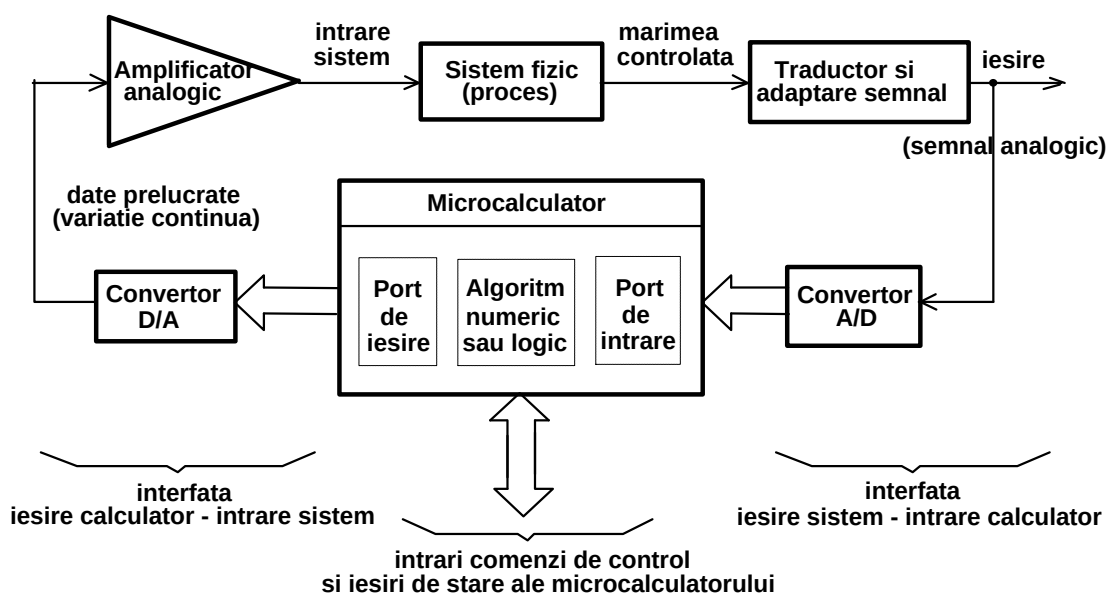


Figura 1.2. Structura de principiu a unui sistem de control numeric

În instalațiile conduse cu sisteme microprocesor, de acest tip, operațiile se desfășoară continuu, ciclic, sistemul de calcul efectuând operații de măsurare a datelor, calcule, și dând comenzi în mod succesiv. Asemenea aplicații ne vor interesa în cadrul cursului de față, cu precădere. Ele sunt denumite **aplicații în timp real**, și au specific faptul că echipamentul controlează funcționarea unui proces, calculele făcându-se în timpul controlării acestei funcționări. În esență, orice aplicație în timp real se va desfășura ca în figura 1.3.

În mod corespunzător, secvențializarea operațiilor din sistem va determina o succesiune a operațiilor ca în figura 1.4.

Trebuie remarcată deosebirea fundamentală față de sistemele de calcul "clasice", utilizate doar pentru calcule, în care desfășurarea acestora nu este legată și nici condiționată în vre-un fel de evenimente exterioare (evoluția procesului condus, nu trebuie să scape de sub controlul

sistemului numeric, in cazul aplicatiilor in timp real). Evident, toate aceste aspecte vor implica aparitia unor abordari si tratari specifice ale problemelor ce apar, pentru sistemele de acest gen (pentru controlul proceselor).

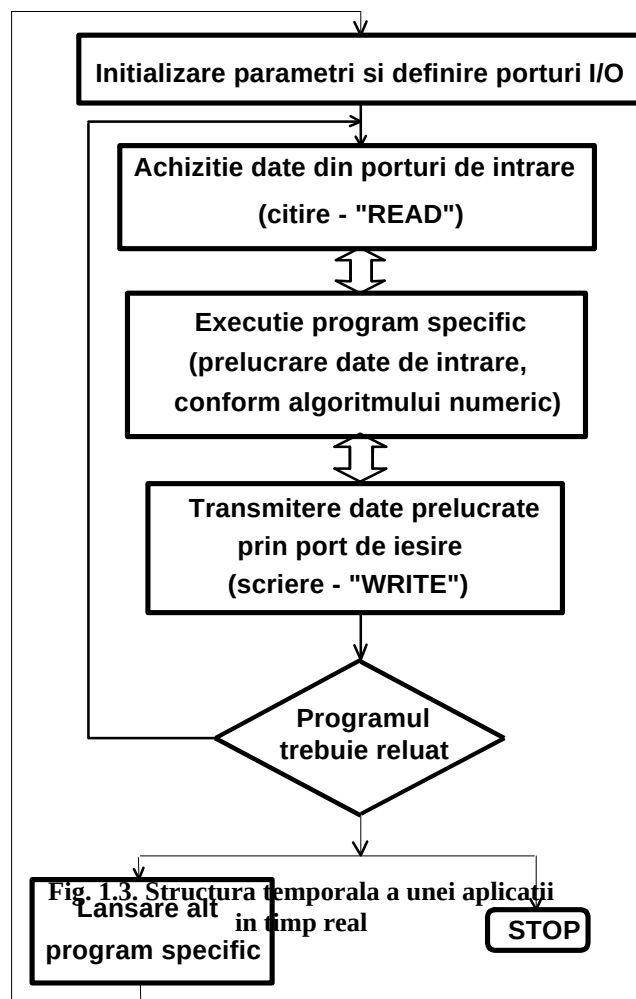


Fig. 1.3. Structura temporală a unei aplicații
Lansare alt
program specific

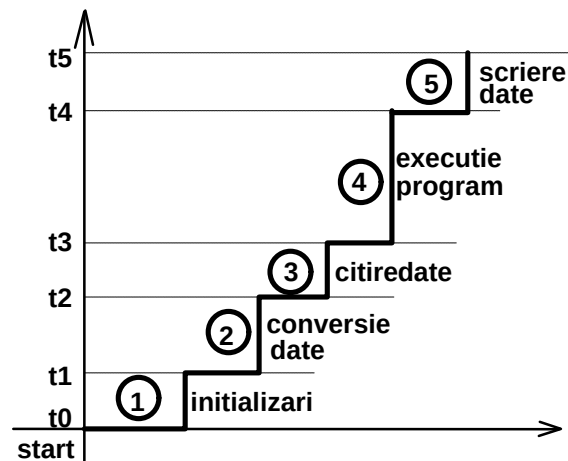


Fig. 1.4. Secventializarea operatiilor intr-o aplicatie de timp real

Ca si in alte domenii ale proiectarii ingineresti, fazele metodologice si de dezvoltare ale unui asemenea sistem sunt dictate cu precadere de catre specificatiile referitoare la performantele si caracteristicile produsului final. Pentru sistemele cu microprocesoare, un element caracteristic este reprezentat de catre cele doua elemente interconditionate - proiectarea structurii constructive - **hardware**, si proiectarea programelor - **software**. Datorita legaturii foarte stranse

dintre structurile hardware si software, cele doua componente ale proiectarii interactioneaza permanent. Astfel, conceperea acestor sisteme este ceva mai complexa decat in cazul sistemelor logice "clasice". Activitatile cheie si fazele de dezvoltare ale unui produs bazat pe utilizarea unui microprocesor sunt prezentate in figura 1.5.

Prezentul volum va incerca acoperirea principalelor aspecte legate de conceptele, notiunile, structura si caracteristicile functionale ale sistemelor microprocesor, atat in general cat si prin concretizare pentru anumite tipuri de microprocesoare si circuitele aferente. Se va insista pe aspectele concrete, aplicative, si se vor exemplifica unele implementari de structuri microcalculator de proces, pentru aplicatii in echipamentele electrice, dupa cum s-a mentionat.

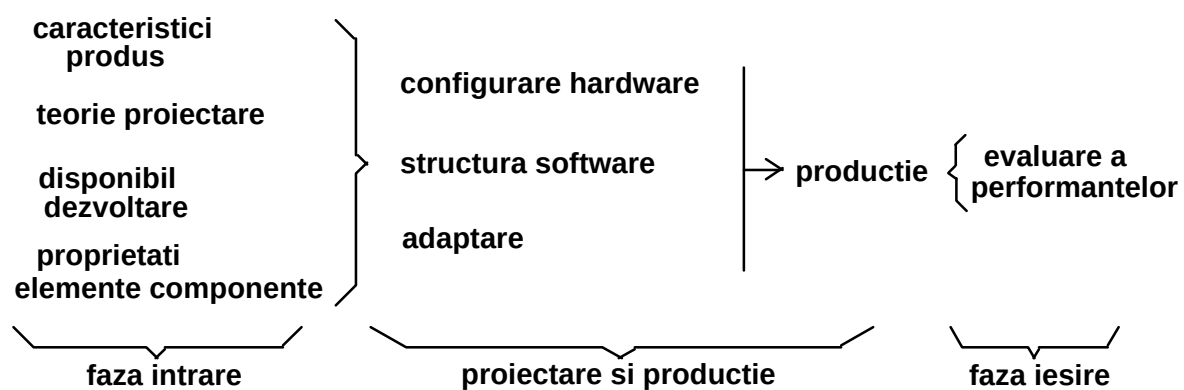


Fig. 1.5. Etapele proiectarii si producerii unui sistem cu microprocesor

2. NOTIUNI INTRODUCTIVE

În prezentarea sistemelor cu microprocesoare, anumite elemente de bază sunt folosite cu precădere, și este necesară cunoașterea lor, ca un punct de pornire, ca un bagaj minim de cunoștințe, pentru înțelegerea aspectelor legate de aceste sisteme (structura, funcționalitate, principii de proiectare și realizare). În acest sens, două categorii de noțiuni de bază trebuie cunoscute, și anume *elemente de aritmetică calculatoarelor*, și *elemente de logică*, corelate cu tipurile uzuale de circuite logice TTL (circuite integrate SSI¹ - porți, bistabile, etc., cât și circuite integrate MSI² - registre, multiplexoare, decodificatoare, circuite PLD³, etc.).

Prezentul paragraf va face o trecere în revistă a celor mai importante aspecte relativ la aceste elemente.

2.1 ARITMETICA NUMERELOR ÎNTREGI

Un sistem de calcul cu microprocesor trebuie să aibă anumite capacități de calcul, putând să implementeze cele patru operații aritmetice fundamentale, de adunare, scădere, înmulțire și împărțire. Dacă fiinta omenească efectuează aceste operații în sistemul zecimal de numărare, un microprocesor este prin structură și o mașină binară, toate informațiile sunt memorate în cod binar, iar operațiile ce se desfășoară în ea se efectuează de asemenea în sistemul binar de numărare. De asemenea, sistemele hexazecimal și (uneori) octal, sunt utilizate la listarea programelor, astfel încât inginerul ce dorește proiectarea și/sau utilizarea unei asemenea mașini trebuie să fie familiarizat cu o varietate de sisteme de numărare. Totodată, interfatarea cu utilizatorul uman presupune adeseori preluarea de date în format zecimal, (tastaturi, console) și, după prelucrare, prezentarea rezultatelor de asemenea zecimal (afisaje, listinguri, etc.). Astfel, sunt necesare uneori operații în cod BCD în interiorul microprocesorului, și deci sunt necesare tehnici de conversie binar-BCD și reciproc.

Sub o formă generală, reamintim că un număr reprezentat în baza b de numărare, având n cifre supraunitare ($a_{n-1}, a_{n-2}, \dots, a_0$), și m cifre subunitare ($a_{-1}, a_{-2}, \dots, a_{-m}$), va avea valoarea în baza 10 dată de relația

$$(N)_{10} = a_{n-1} \times b^{n-1} + a_{n-2} \times b^{n-2} + \dots + a_1 \times b^1 + a_0 \times b^0 + a_{-1} \times b^{-1} + \dots + a_{-m} \times b^{-m}$$

¹ SSI - Simple Scale Integrated = integrate pe scară simplă

² MSI - Medium Scale Integrated = integrate pe scară medie

³ PLD - Programmable Logic Device = circuit logic programabil

Astfel, de exemplu, numarul binar $(1011.11)_2$ va avea in baza 10 valoarea

$$(N)_{10} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = 8 + 0 + 2 + 0.5 + 0.25 = 11.75_{10}$$

Deoarece toate masinile de calcul existente opereaza cu elemente de circuit cu doua stari (de tipul tranzistor in stare saturat sau blocat, avand la iesire doua nivele de tensiune continua distincte - tipic 0V, respectiv 5V), sistemul binar de reprezentare a informatiei este unicul prezent in toate aceste sisteme de calcul). Starea de saturare a tranzistorului ($0 \div 0.4$) V, este asociata valorii binare 0, iar starea de blocare ($2.4 \div 5$) V, este asociata valorii binare 1.

Metodologia practica a conversiei numerelor dintr-o baza de numarare in alta este presupusa cunoscuta, si nu se mai prezinta aici. Trebuie totusi mentionate ca fiind deosebit de importante cazurile, uzual utilizate in cadrul sistemelor numerice, dupa cum s-a mentionat, de conversie directa intre sistemele binar si octal, respectiv binar - hexazecimal. Aceste conversii sunt prezentate tabelat in figurile 2.1, si respectiv 2.2.

Octal	Binari
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Fig.2.1. Conversia octal - binar

Hex	Binari
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Fig. 2.2. Conversia hexazecimal - binar

Hex	Binari
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Conversia dintr-o baza intr-alta in acest mod se va face utilizand direct reprezentarile din cele doua tabele. Astfel, de exemplu:

(110 001 011 100) ₂
↓ ↓ ↓ ↓
(6 1 3 4) ₈

Conversia binar ® octal se efectueaza prin gruparea a cate trei cifre binare, incepand din dreapta numarului, si inlocuire cu echivalentul octal corespunzator.

(4 3 2 7) ₈
↓ ↓ ↓ ↓
(100 011 010 111) ₂

Reciproc, pentru *conversia octal ® binar*, se procedeaza in mod similar:

Se inlocuieste fiecare cifra octala cu cei trei biti din codul binar corespunzator.

La fel se procedeaza si pentru *conversiile hexazecimal « binar*:

(1011 1010 0011 0010) ₂		(4 F C 2) ₁₆
↓ ↓ ↓ ↓	respectiv	↓ ↓ ↓ ↓
(B A 3 2) ₁₆		(0100 1111 1100 0010) ₂

În interiorul masinilor numerice, numerele sunt reprezentate prin grupe de cifre binare. Aceste grupe de cifre binare sunt numite **cuvinte**. Lungimea cuvintelor pe care le poate prelucra un microprocesor constituie o caracteristica importanta a acestuia, reprezentand, dupa cum s-a mai precizat, un indiciu pretios in aprecierea capacitatii de calcul a sistemului. Sunt tipice **cuvinte** cu o lungime de 4, 8, 16 sau 32 de biti. **Un cuvânt de 8 biti** se mai numeste si **octet** sau, in terminologia tehnica consacrata, **byte**. Cuvintele microprocesoarelor sunt memorate in registre. Un registru poate fi implementat cu un sir de circuite bistabile, fiecare dintre acestea putand fi considerat ca un bit de memorie. Din punct de vedere functional, o masina numerica poate fi privita ca un sistem format din registre, iar datele sunt transferate dintr-un registru in altul in cursul efectuării operatiilor aritmetice si logice. Acesta este evident un mod simplificat de a prezenta lucrurile, dar poate da o imagine de baza asupra a ceea ce se intampla.

În ceea ce priveste operatiile aritmetice de baza, acestea se bazeaza pe regulile specifice, caracteristice operatiilor respective. Regulile depind de tipul numerelor: întregi fara semn; întregi cu semn; numere in reprezentare *BCD*; numere fractionare.

Nu este locul si nici cazul de a prezenta algoritmi specifici de implementare a acestor operatii, pentru fiecare caz in parte. Se vor face doar unele consideratii si remarci, legate de specificul implementarii acestor operatii in sistemele numerice.

2.1.1 Aritmetica numerelor întregi, pozitive (fara semn).

În cazul *adunarii* (sau *scaderii*, echivalenta în cazul reprezentării numerelor în complement fata de 2), trebuie remarcata problema aparitiei *depasirilor* (*overflow*), sau a *imprumuturilor* (*borrow*), daca suma sau diferenta nu este reprezentabila într-un cuvânt de date (de exemplu, suma a doua numere reprezentate pe 8 biti, care adunate dau un rezultat peste 256, va necesita 9 biti pentru reprezentare corecta). Data fiind lungimea finita a cuvintelor de date în microprocesor, operatia de adunare sau de scadere va trebui sa semnalizeze suplimentar efectuării calculului, aparitia acestui bit de depasire; altfel, daca rezultatul este considerat pe 8 biti, se pierde chiar bitul cel mai semnificativ, alterandu-se rezultatele si obtinandu-se cu totul alte valori decat cele corecte. De exemplu, o asemenea situatie apare în cazul urmator:

	2 ⁷	2 ⁶	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰
	1	1	0	1	1	0	1	1 +
	0	1	0	0	0	1	1	0
1	0	0	1	0	0	0	0	1
↑								
Depasire								

O situatie similara poate aparea în cazul scaderii, daca descazutul este mai mare decat scazatorul (apare un bit de imprumut).

În ceea ce priveste *înmultirea*, trebuie remarcat ca rezultatul înmultirii a doua numere reprezentate pe cate un cuvânt de date se va reprezenta pe doua cuvinte de date. În general,

inmultirea unui numar reprezentat pe m biti cu un numar reprezentat pe n biti (intregi, fara bit de semn), conduce la obtinerea unui rezultat reprezentat pe $(m+n)$ biti. O situatie aparte va aparea la impartirea a doua numere, cand trebuie sesizata si semnalizata incercarea de impartire la zero, al carei rezultat nedeterminat poate conduce la efecte imprevizibile asupra comportarii sistemului.

Trebuie remarcat ca practic toate microprocesoarele au implementate in setul de instructiuni operatiile de adunare si de scadere. Doar unele microprocesoare au in setul de instructiuni operatiile de inmultire (si eventual, de impartire). In cazul absentei acestora din setul de instructiuni al microprocesorului respectiv, implementarea lor se face prin program, pe baza operatiilor elementare de adunare si a celor logice.

Ca o remarca generala, *cuvintele de date de 8 biti* vor putea contine *valori numerice pozitive intre 0 si 255 ($2^8 - 1$)*. *Cuvintele de date pe 16 biti* vor putea contine *valori intre 0 si 65535 ($2^{16} - 1$)*.

2.1.2 Aritmetica numerelor intregi, cu semn

O posibilitate de distingere a numerelor pozitive de cele negative intr-un calculator, este de a folosi un bit suplimentar, pentru codificarea semnului. Prin conventie, bitul cel mai semnificativ (din stanga), este asociat semnului numarului: daca acest bit este 0, numarul este considerat pozitiv; daca este 1, numarul este negativ. Evident, in aceste cazuri, pentru cuvinte de date cu lungime data, plaja de valori numerice ce pot fi reprezentate difera de cazul numerelor intregi fara bit de semn. Astfel, intr-un *octet* se vor putea reprezenta *numere intregi intre -128 si +127*.

Cea mai simpla posibilitate de reprezentare a numerelor cu semn, care este insa practic nefolosita, deoarece ingreuneaza implementarea operatiilor de calcul aritmetice, este cea care utilizeaza bitul de semn si modulul numarului. Astfel, de exemplu in aceasta reprezentare:

$$\begin{aligned} -23_{10} &= 1, 0010111_2 \\ \text{iar} \\ +23_{10} &= 0, 0010111_2 \end{aligned}$$

Practic in totalitatea cazurilor se utilizeaza reprezentarea numerelor cu semn in forma complementara, ceea ce favorizeaza in mod decisiv implementarea hardware a operatiilor aritmetice cu aceste numere. Exista doua tipuri de reprezentare:

- (1) ***complement fata de 2 (2's complement)***;
- (2) ***complement fata de 1 (1's complement)***.

Deoarece reprezentarea in complement fata de 1 ridica anumite probleme, procesarile cu numere cu semn se fac cu numere reprezentate in complement fata de 2. Calculul complementului fata de doi, pentru un anumit numar X , reprezentat pe n biti, va fi:

$$[X]_2 = 2^n - X$$

Astfel, prin reprezentarea complementului fata de doi al unui numar, se obtine reprezentarea binara a opusului acestui numar (deci $[X]_2 = -X$). De exemplu, fie $X = 6_{10} = 0110_2$, $n=4$. Calculand complementul fata de doi al acestui numar, se obtine reprezentarea pentru $-X = -6_{10}$, data de

$$[X]_2 = 2^4 - 0110 = 10000 - 0110 = 1010$$

Practic, se utilizeaza urmatoarele *metode*, pentru obtinerea complementului fata de 2 al unui numar:

(a) *se inverseaza toti bitii numarului, si se aduna 1 la cel mai putin semnificativ bit.*

De exemplu, fie: $X = 0110$
 X inversat va fi 1001
 se aduna 1 1
 Rezultat: $[X]_2 = 1010$

(b) *se lasa neschimbate cifrele din numar pana la primul 1, inclusiv (de la dreapta la stanga), inversandu-se celelalte cifre.*

De exemplu, fie $X = 0110 \Rightarrow [X]_2 = 1010$

$\begin{array}{cc} \overline{} & \overline{} \\ \swarrow & \searrow \end{array}$	$\begin{array}{cc} \overline{} & \overline{} \\ \swarrow & \searrow \end{array}$
se inverseaza	raman nemodificate

In aceste conditii, un numar reprezentat pe un cuvânt de lungime data în complement fata de 2, va avea implicit 1 bit de semn (bitul cel mai semnificativ - bitul din stanga), si reprezentarea propriu-zisa pe ceilalti biti. Astfel,

$+19 = 0, 0010011$
 $\swarrow$ $\searrow$
 bit de semn $\searrow$ valoarea $X > 0$
 $\swarrow$ $\searrow$
 $-19 = 1, 1101101$ $\swarrow$ 2's complement al valorii $X < 0$

De remarcat ca plaja valorilor astfel reprezentabile (pe un octet de la -128 la +127) este nesimetrica.

Adunarea si scaderea în aceasta reprezentare vor da întotdeauna rezultate corecte, cu conditia evidenta de a nu aparea depasiri ale limitelor de reprezentare (**overflow** = *depasire superioara*, sau **borrow** = *imprumut*). În ceea ce priveste înmultirea, se prefera convertirea numerelor la valoarea pozitiva (modul), înmultirea între numerele pozitive, si corectia finala în functie de semnul rezultatului, dedus separat din analiza semnelor celor doi termeni.

2.1.3 Aritmetica numerelor zecimale codificate binar (BCD)

Daca se utilizeaza frecvent numerele in reprezentare zecimala - si corespunzator BCD, la citirea/scrierea numerelor dinspre / spre exteriorul sistemului, implementarea operatiilor aritmetice cu numere in reprezentare BCD nu este usor de realizat. Complicatiile pe care le presupune implementarea acestor operatii conduc de obicei la preferarea urmatoarei scheme de lucru: *reprezentare BCD* ® *reprezentare binara propriu-zisa* ® *calcule* ® *conversie in reprezentarea BCD a rezultatului*.

2.1.4 Aritmetica numerelor fractionare

O valoare fractionara subunitara poate fi reprezentata in cuvantul de calculator prin simpla mutare a virgulei la extremitatea din stanga a cuvantului de date. Aceasta este echivalent, de exemplu pentru un cuvant de 8 biti, cu impartirea numarului intreg reprezentat pe cuvantul de 8 biti, prin 256. Astfel, numarul continut de reprezentarea respectiva, va fi $X = 0 . x_7 x_6 \dots x_0$. Valoarea zecimala a numarului este $[X]_{10} = x_7 2^{-1} + x_6 2^{-2} + \dots + x_0 2^{-8}$. Daca numarul contine doar cifre de 0, valoarea corespunzatoare este $N = 0.00000000 = 0/256 = 0$. Daca toti bitii x_7 la x_0 sunt 1, valoarea corespunzatoare este $N = 0.11111111 = 255/256 = 0.99609375$.

Consideratii similare cazului numerelor pozitive se vor face pentru reprezentarea numerelor cu semn, utilizand reprezentarea in cod complementar fata de doi.

O remarca speciala trebuie facuta in ceea ce priveste precizia reprezentarii numerelor fractionare. Cu exceptia cazurilor cand valoarea fractionara poate fi reprezentata exact pe cuvantul de date, uzual nu se poate reprezenta o marime fractionara exact pe un numar dat de biti (exista, de exemplu, doar 256 de valori numerice intre 0.0 si 1.0, reprezentabile exact pe 8 biti). In rest, reprezentarea valorilor fractionare se va face prin introducerea unei *erori*, care poate fi de *trunchiere* (se neglijeaza bitii suplimentari din reprezentarea cu precizie peste lungimea cuvantului de date), sau de *rotunjire* (se ajusteaza numarul, in functie de valoarea partii eliminate, la valoarea cea mai apropiata de reprezentarea exacta).

Evident, operatiile aritmetice cu numere fractionare vor implica erori suplimentare, de calcul. In cazul inmultirilor, trebuie remarcat ca rezultatul va fi intotdeauna subunitar; reprezentarea rezultatului pe un cuvant de date va conduce insa intotdeauna la o eroare de trunchiere sau de rotunjire a rezultatului.

2.2 ELEMENTE DE LOGICA CIRCUITELOR NUMERICE

In afara capacitatii de a efectua operatii aritmetice, un microprocesor poate fi programat sa implementeze operatii logice ca **AND**, **OR**, **XOR**, **NOT**, etc. Totodata, elemente logice (*porti*), sunt utilizate ca parti componente ale sistemului microcalculator. Vom prezenta in cele ce urmeaza o trecere in revista a principiilor logicii booleene ce sta la baza regulilor de functionare a acestor circuite, cat si comportarea celor mai utilizate elemente logice intalnite uzual intr-un sistem microcalculator.

2.2.1 Logica booleana

Iata in primul rand rememorate relatiile de echivalenta ale algebrei booleene, care stau la baza analizei / proiectarii functionarii acestor circuite si / sau operatii logice:

(a) Postulate

$$\begin{aligned} \Rightarrow \bar{x} &= 0 \text{ daca } x \neq 1 ; & x &= 1 \text{ daca } x \neq 0 \\ \Rightarrow \bar{0} &= 1 ; & \bar{1} &= 0 \\ \Rightarrow 0 \cdot 0 &= 0 ; & 1 + 1 &= 1 \\ \Rightarrow 1 \cdot 1 &= 1 ; & 0 + 0 &= 0 \\ \Rightarrow 1 \cdot 0 &= 0 ; & 0 + 1 &= 1 + 0 = 1 \end{aligned}$$

(b) Teoreme cu o singura variabila

$$\begin{aligned} \Rightarrow x + 1 &= 1 ; & x \cdot 0 &= 0 \text{ (element nul)} \\ \Rightarrow x + 0 &= x ; & x \cdot 1 &= x \text{ (identitate)} \\ \Rightarrow x + x &= x ; & x \cdot x &= x \text{ (idempotenta)} \\ \Rightarrow x + \bar{x} &= 1 ; & \bar{x} \cdot x &= 0 \text{ (complement)} \\ \Rightarrow \overline{\bar{x}} &= x \text{ (involutie)} \end{aligned}$$

(c) Teoreme cu 2 sau 3 variabile

$$\begin{aligned} \Rightarrow x + y &= y + x ; & x \cdot y &= y \cdot x \text{ (comutativitate)} \\ \Rightarrow x + (x \cdot y) &= x ; & x \cdot (x + y) &= x \text{ (absorbție)} \\ \Rightarrow (x + \bar{y}) \cdot y &= x \cdot y ; & (x \cdot \bar{y}) + y &= x + y \\ \Rightarrow x + y + z &= x + (y + z) = (x + y) + z ; \\ x \cdot y \cdot z &= (x \cdot y) \cdot z = x \cdot (y \cdot z) \text{ (asociativitate)} \\ \Rightarrow (x \cdot y) + (x \cdot z) &= x \cdot (y + z) \end{aligned}$$

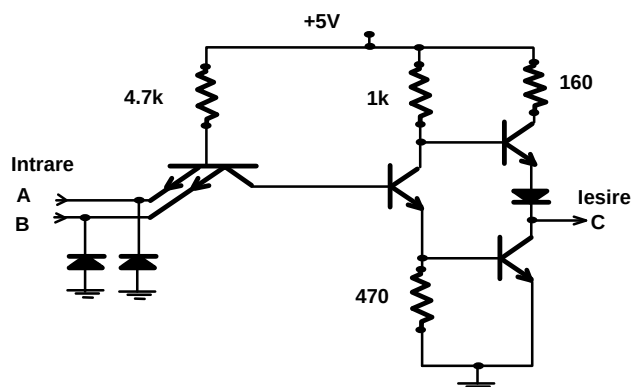
$$\begin{aligned} \Rightarrow (x + y) \cdot (\bar{x} + z) \cdot (y + z) &= (x + y) \cdot (\bar{x} + z) ; \\ (x + y) \cdot (x + z) &= x + (y \cdot z) \text{ (distributivitate)} \\ \Rightarrow (x + y) \cdot (\bar{x} + z) &= (x \cdot z) + (\bar{x} \cdot y) ; \\ (x \cdot y) + (\bar{x} \cdot z) + (y \cdot z) &= (x \cdot y) + (\bar{x} \cdot z) \text{ (consens)} \end{aligned}$$

(d) Teoreme cu n variabile

$$\begin{aligned} \Rightarrow \overline{(x + y + z + \dots)} &= \bar{x} \cdot \bar{y} \cdot \bar{z} \cdot \dots ; \\ (\bar{x} \cdot \bar{y} \cdot \bar{z} \cdot \dots) &= \overline{x + y + z + \dots} \text{ (Teoremele De Morgan)} \\ \Rightarrow f(x_1, x_2, \dots, x_n) &= [x_1 \cdot f(1, x_2, \dots, x_n)] + [\bar{x}_1 \cdot f(0, x_2, \dots, x_n)] ; \\ f(x_1, x_2, \dots, x_n) &= [x_1 + f(0, x_2, \dots, x_n)] \cdot [\bar{x}_1 + f(1, x_2, \dots, x_n)] \\ &\quad \text{(teoremele expansiunii)} \end{aligned}$$

2.2.2 Porti logice

Figura 2.3 prezinta structura unei porti *TTL* (Tranzistor Tranzistor Logic) ce implementeaza o functie NAND cu 2 intrari.



logic 1: (2.4 , 5) V
logic 0: (0 , 0.4) V

Intrare		Iesire
A	B	C
1	1	0
1	0	1
0	1	1
0	0	1

Tabela logica a portii TTL

Fig.2.3. Structura si tabela logica a unei porti TTL de tip NAND cu doua intrari

Similar se vor implementa si alte functii logice elementare. Cele mai utilizate porti *TTL* sunt: *AND*, *OR*, *NAND*, *NOR*, *XOR*, si *Ex NOR*. Figura 2.4 prezinta tabelele de adevar ale celor 6 tipuri de porti (se considera cazul a cate 2 intrari pe poarta) (fig.2.4.a), si simbolurile asociate (fig.2.4.b). Cercurile mici, ce apar la iesirea portilor *NAND*, *NOR*, si *NOR exclusiv* reprezinta negatii, indicand de exemplu, ca o poarta *NAND* poate fi considerata ca o poarta *AND*, a carei iesire este negata (inversata).

A	B	AND	OR	NAND	NOR	XOR	Ex-NOR
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

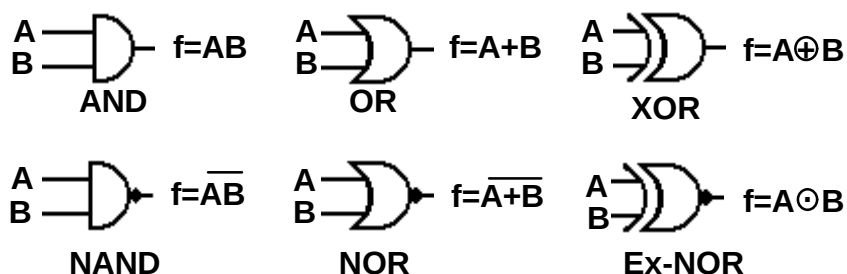


Fig.2.4. Porti logice simple uzuale: (a) tabele de adevar (b) simboluri

2.2.3 Inversoare

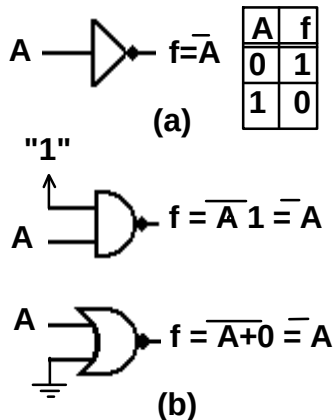


Fig.2.5. Poarta inversoare

Sunt circuite cu schema si diagrama de functionare prezentate in figura 2.5.a.

Este posibila obtinerea unui inversor si prin utilizarea unor porti de tip *NAND* sau *NOR*, ca in figura 2.5.b. Trebuie remarcat ca legarea pinilor portii *NAND* la "1", respectiv pentru poarta *NOR* la "0", este obligatorie, fiind interzisa lasarea neconectata a acestor terminale (caz in care pot aparea fenomene de functionare aleatoare a portilor). O alta posibila solutie este si aceea de a conecta semnalul de intrare la ambii pini de intrare ai uneia dintre cele doua porti (*NAND* sau *NOR*), ceea ce conduce insa la cresterea (dublarea) incarcarii pe semnalul de intrare.

2.2.4 Reprezentari logice alternative

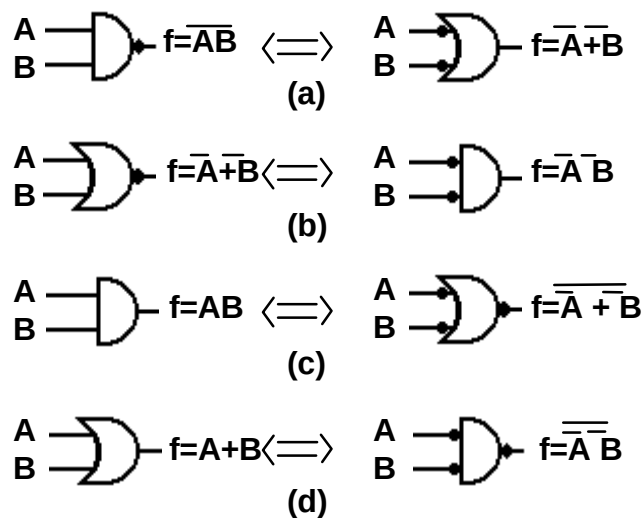


Fig 2.6. Reprezentari logice alternative pentru portile logice

Se pot obtine reprezentari alternative echivalente ale portilor logice, prin aplicarea teoremelor *De Morgan* functiilor implementate de catre aceste circuite. Astfel, *ecuatia unui circuit NAND* va fi

$$f = \overline{A \cdot B} = \overline{A} + \overline{B},$$

deci poarta *NAND* se poate echivala cu o poarta *OR*, avand intrarile negate (v.fig.2.6.a). Similar, pentru o functie *NOR* se va scrie

$$f = \overline{A + B} = \overline{A} \cdot \overline{B},$$

deci ca o functie *AND* cu intrarile negate (v.fig. 2.6.b). Relatii similare se pot obtine si pentru portile *AND*, *OR*, etc.

Se poate utiliza oricare dintre reprezentarile uneia dintre porti, in schema unui circuit. Totusi, cele doua reprezentari ale unei porti au interpretari diferite. Astfel, pentru poarta *NAND* de exemplu (fig. 2.6.a), in **prima reprezentare**, se poate spune ca iesirea devine de nivel logic 0, cand toate intrarile sunt la nivel logic 1; cea de-a **doua reprezentare** sugereaza ca iesirea devine 1, daca oricare dintre intrari este la nivel logic 0. Prima reprezentare se va utiliza *cand iesirea este uzual la nivel 1, dar devine activa pe nivel logic 0*. Se spune ca **iesirea este activa pe nivel logic 0 (active low)**. Reprezentarea alternata va fi utilizata **cand starea de activare a portii este 1 logic la iesire (active high)**. Similar se vor interpreta si celelalte porti si reprezentarile lor echivalente.

2.2.5 Poarta tri-state

Poarta **tri-state** (cu *trei stari*) reprezinta o componenta importanta in sistemele microprocesor. Exista atat tipuri de circuite *tri-state inversoare*, cat si *neinversoare*. Figura 2.7 prezinta tabela de adevar si reprezentarea simbolica a circuitului tri-state (*neinversor* in fig. 2.7.a, *inversor* in fig. 2.7.b).

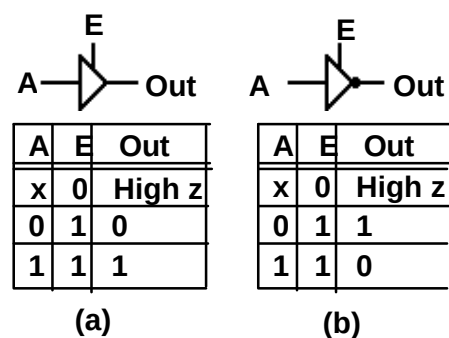
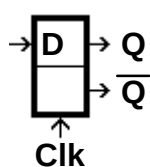


Fig.2.7. Poarta tri-state neinversoare (a) si inversoare (b).

Utilizarea multor elemente de circuit cu iesiri tri-state este datorata posibilitatii de a avea starea de mare impedanta la iesirea circuitului. Aceasta permite decuplarea semnalului de iesire de intrarea circuitului. Astfel, iesirile mai multor circuite tri-state pot fi conectate impreuna, nivelul logic pe iesirea comuna fiind dictat de circuitul activat (cu semnalul de selectie $E=1$) la un moment-dat. Detalii suplimentare referitor la functionarea circuitelor tri-state si utilizarea acestora in sistemele cu microprocesoare se vor prezenta in capitolul urmator.

2.2.6 Bistabilul de tip D

Bistabilul D este cel mai utilizat in sistemele cu microprocesoare ca element de memorie de 1 bit (mai rar se utilizeaza si bistabile de tip T, SR, sau JK). Circuitul are doua stari stabile, avand reprezentarea conventionala si tabela de adevar din figura 2.8. Se remarca faptul ca



D^t	Q^t	$Q^{t+\delta t}$
0	0	0
0	1	0
1	0	1
1	1	1

Fig. 2.8. Bistabilul D

$$Q^{t+\delta t} = (D \cdot \bar{Q} + \bar{D} \cdot Q)^t = D^t,$$

ceea ce reprezinta **ecuatia caracteristica** a bistabilului D, indicand faptul ca iesirea de la momentul $t+\delta t$ urmareste valoarea intrarii D la momentul t . Transferul este controlat de catre semnalul **Clk**. Exista doua categorii de bistabile D, din punctul de vedere al controlului

transferului datei prin intermediul semnalului **Clk**.

Un *prim tip* este bistabilul D **activ pe front**. In acest caz, valoarea semnalului de la intrarea D este transmisa la iesirea Q, la trecerea semnalului Clk din starea 1 in starea 0 (ca in figura 2.9.a), deci pentru front descrescator, sau la trecerea semnalului Clk din starea 0 in starea 1 (ca in figura 2.9.b), deci pentru front crescator (se remarca din figura 2.9 si simbolurile utilizate in aceste cazuri pentru circuitul respectiv).

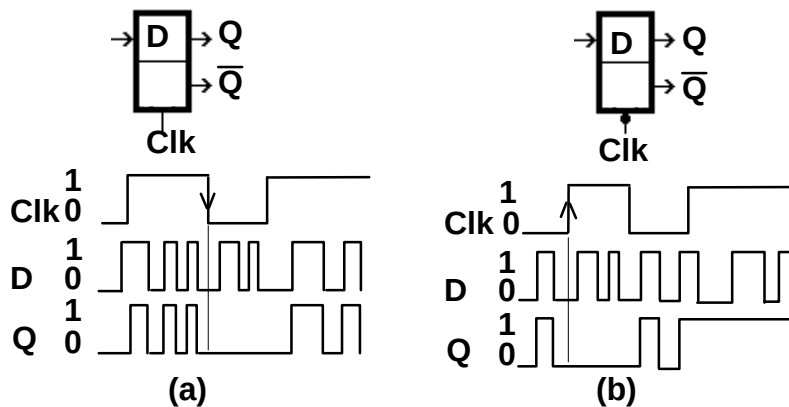


Fig.2.9. Bistabilul D cu inscriere pe front.

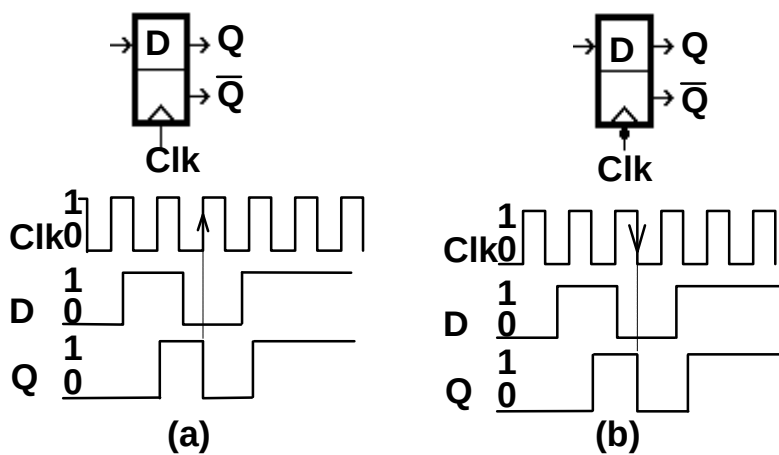


Fig.2.10. Bistabilul D cu inscriere pe nivel.

Un *alt tip* este bistabilul *D activ pe nivel*. In acest caz, pe perioada nivelului activ al semnalului *Clk* (1 in cazul din figura 2.10.a, respectiv 0 in cazul din figura 2.10.b), iesirea *Q* urmareste variatiile intrarii *D*. La dezactivarea semnalului *Clk*, iesirea *Q* ramane in ultima stare in care a fost (vezi figura 2.10). A se remarca din nou simbolurile utilizate pentru circuite in aceste cazuri.

Mai trebuie remarcata prezenta uzuala a inca 2 semnale de comanda ale bistabilului *D*, asincrone, servind la setarea sau resetarea circuitului. Sunt semnale asincrone, uzual active pe nivel 0. Figura 2.11 prezinta simbolul bistabilului si tabela de adevar in acest caz. De remarcat ca este interzisa valoarea 0, simultana, a celor doua semnale $\bar{S}$ (set - inscriere) si $\bar{R}$ (reset - stergere).

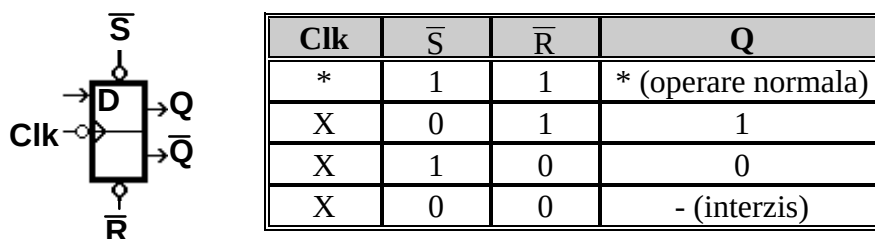


Fig. 2.11. Utilizarea semnalelor asincrone de Set si Reset.

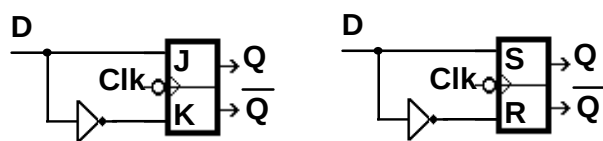


Fig.2.12. Conversia bistabilelor JK sau RS in bistabile D

Bistabilele de alte tipuri se pot converti usor la o comportare similara bistabilului *D*. Figura 2.12 prezinta modul de conversie al bistabilelor de tip *JK*, respectiv *RS*, in bistabile de tip *D*.

In ceea ce priveste functionarea bistabilului, trebuiesc respectate anumite restrictii la evolutia in timp a semnalelor *D* si *Clk*. Astfel, datele trebuiesc sa fie stabile cu cel putin un interval de timp t_{su} (v.figura 2.13), inainte de tranzitia semnalului de ceas. t_{su} se numeste timp de stabilire (*setup time*), si uzual are valori cuprinse intre 5 si 50 de nanosecunde.

De asemenea, datele trebuie sa fie stabile cel putin o perioada t_h (timp de mentinere -*hold time*), dupa tranzitia semnalului de ceas (uzual intre 0 si 10 ns), v.fig.2.13. De multe ori, se specifica in catalogul produsului, si latimea minima a semnalului de ceas (t_w - pulse width), si o frecventa maxima a ceasului, f_m . De remarcat in figura 2.13, ca intervalele de timp se masoara de la 0.5 din amplitudinea semnalelor.

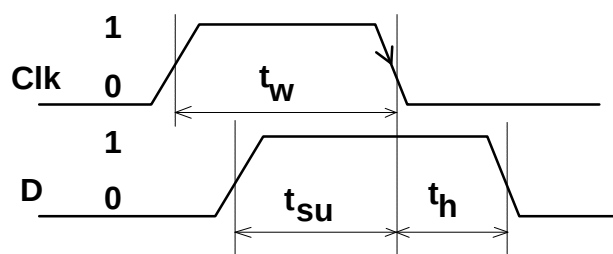


Fig. 2.13. Caracteristici temporale ale semnalelor de intrare ale bistabilului

Bineinteles, exista intotdeauna o intarziere intre tranzitia semnalului de ceas, si modificarea nivelului iesirii, numita intarziere de propagare (*propagation delay*), a bistabilului. Intarzierile respective (la tranzitii ale *Q* de la 0 la 1, respectiv de la 1 la 0), sunt date de catalog ale circuitului, la fel ca si cele referitoare la semnalele de setare si resetare.

2.2.7 Registre.

Registrele reprezinta o componenta fundamentala in orice microcalculator. Insi microprocesorul contine un numar de registre. Deasemenea, circuitele de memorie pot fi privite ca niste matrici de registre, iar unele dintre porturile de I/E ca registre.

Un registru consta dintr-un sir de bistabile si este utilizat pentru a memora informatie binara (temporar sau permanent). Figura 2.14 prezinta o schema bloc a unui registru. Datele pot fi inscise in registru prin semnalul de control **WRITE**, si raman inscise pana la o inscriere ulterioara. Citirea datelor se va face oricand, prin activarea semnalului de citire (**READ**).

O operatie frecventa intr-un microprocesor este transferul paralel al datelor intre doua registre. Figura 2.15 prezinta schema de principiu a acestei operatii. Incarcate initial in registrul A, datele sunt transferate in registrul B la aparitia semnalului **TRANSFER** pe linia de ceas a registrului B. De remarcat ca continutul registrului A este neschimbat, pana la o modificare ulterioara (stergere, incarcare cu alte date). Simbolic, vom reprezenta transferul continutului registrului A in registrul B prin relatia: $T:(B) \leftarrow (A)$. "La primirea semnalului de

control T, se transfera continutul registrului A in registrul B". Acest tip de registru este de tip **PIPO**¹.

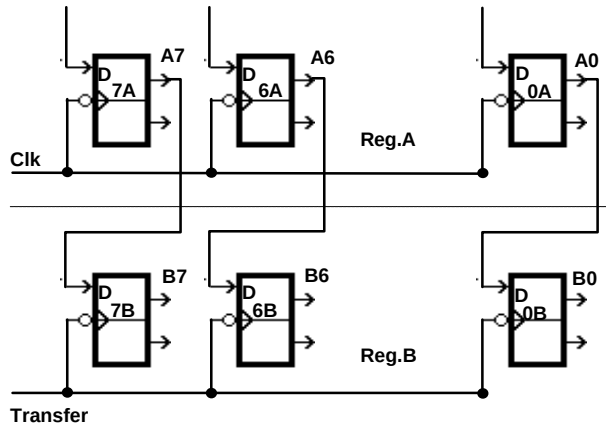


Fig. 2.14 . Schema bloc a unui registru

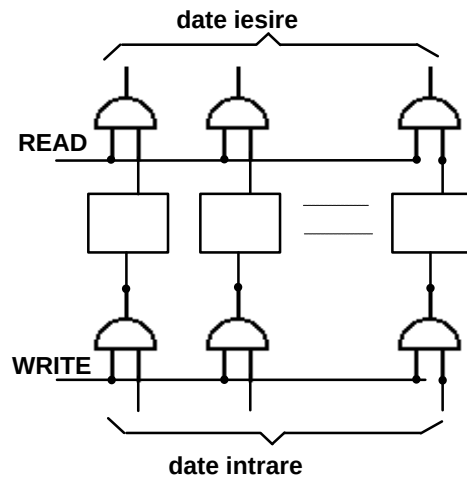


Fig.2.15. Transferul de date între 2 registre

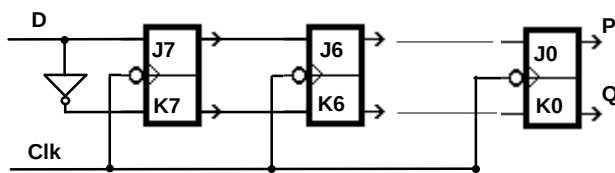


Fig. 2.16. Schema unui registru serial.

O altă operație frecventă în sistemele microprocesor este transferul serial al datelor între registre. De exemplu, figura 2.16 conține un asemenea registru cu transfer serial al datelor (realizat cu bistabile JK). Transferul unui bit de la intrarea D până la ieșirea P a registrului va

necesita în acest caz 8 semnale de ceas. Acest tip de registru se numește **SISO**¹. Figura 2.17 prezintă schema de realizare a transferului serial între două registre de câte 8 biți fiecare. De remarcat că registrul A își păstrează conținutul, prin reciclarea informației între ieșire și intrare. Durata transferului va fi evident de 8 pulsuri de ceas.

¹ PIPO - Paralel Input Paralel Output = intrare paralela, ieșire paralela

¹ SISO - Serial Input Serial Output = intrare serială, ieșire serială

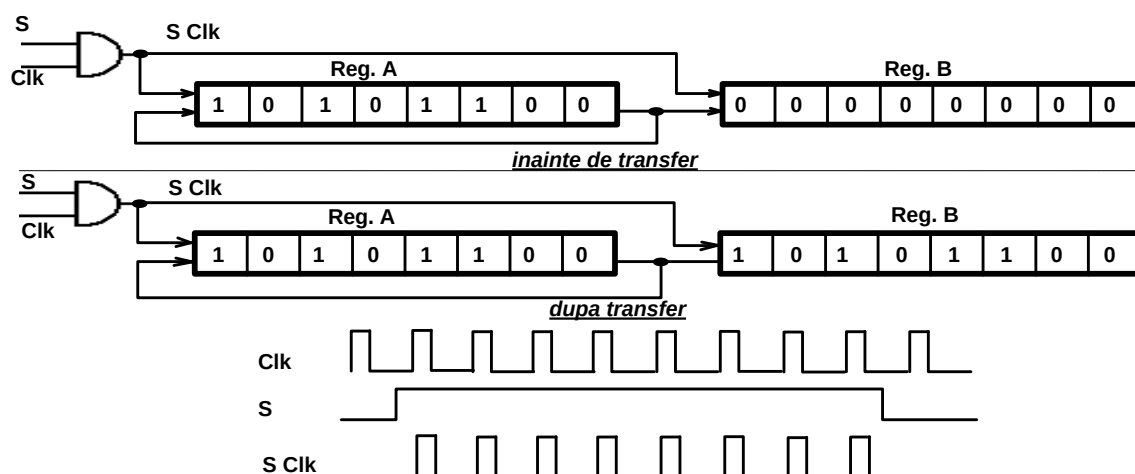


Fig. 2.17 Transferul serial al datelor între două registre

Un tip uzual de registru utilizat în microcalculatoare este **registru de 8 biți de tip latch**¹, care utilizează bistabile D active pe front pozitiv, ca în figura 2.18. Datele de la intrare sunt transferate la ieșirea registrului pe durata nivelului "1" al semnalului **Clk**, rămânând la valorile avute în momentul frontului cazator al semnalului de ceas. Semnalul asincron de ștergere va trece la valoarea "0" toate ieșirile registrului, simultan.

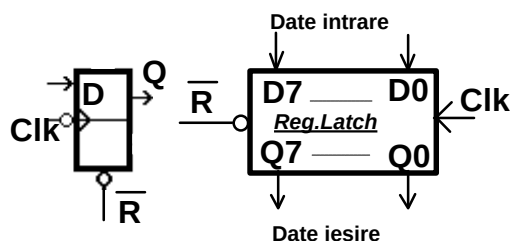


Fig. 2.18. Registru de tip latch.

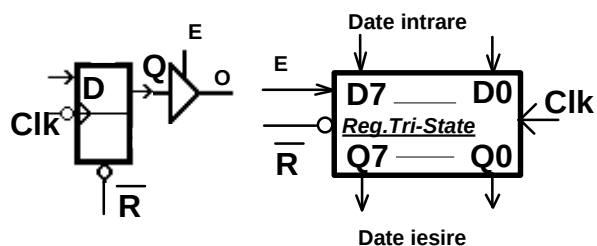


Fig. 2.19. Registru cu ieșiri tri-state

Figura 2.19 prezintă un **registru cu ieșirile tri-state**. Înscrierea datelor se face normal, dar ieșirile **00**, **07** vor conține datele înscrise în registru doar la aplicarea semnalului de activare a circuitelor tri-state de la ieșirea acestuia, prin semnalul de activare **E**.

2.2.8 Multiplexoare

Un multiplexor selectează o ieșire din n intrări (n fiind uzual 4, 8, sau 16). Selecția liniei de ieșire se face cu semnale de control. Figura 2.20 prezintă un asemenea multiplexor cu patru intrări (**D0**, **D1**, **D2** și **D3**), și ieșirea **f** (și negata sa). Liniile de control **A** și **B** și semnalul de activare **E** completează circuitul. Din punct de vedere funcțional, multiplexorul poate fi privit ca un comutator cu mai multe poziții, selectând una dintre ele la un moment dat (**f**, în figura

¹ latch = agata

2.20,b). Implementarea schematica a circuitului este data in figura 2.20.c, obtinut prin sinteza functiei f conform expresiei

$$f = \bar{E} \cdot (\bar{A} \cdot \bar{B} \cdot D_0 + \bar{A} \cdot B \cdot D_1 + A \cdot \bar{B} \cdot D_2 + A \cdot B \cdot D_3)$$

Cand $A=0, B=0 \Rightarrow \bar{A} \cdot \bar{B} = 1$, si se selecteaza linia D_0 ; cand $A=0, B=1 \Rightarrow \bar{A} \cdot B = 1$, si se selecteaza D_1 , etc. Daca $E=1$, se inhiba operatia intregului circuit.

Exista si circuite cu doua sau mai multe multiplexoare. Un circuit multiplexor poate fi privit ca un circuit care convertește informatia paralela de la intrare, într-o forma seriala.

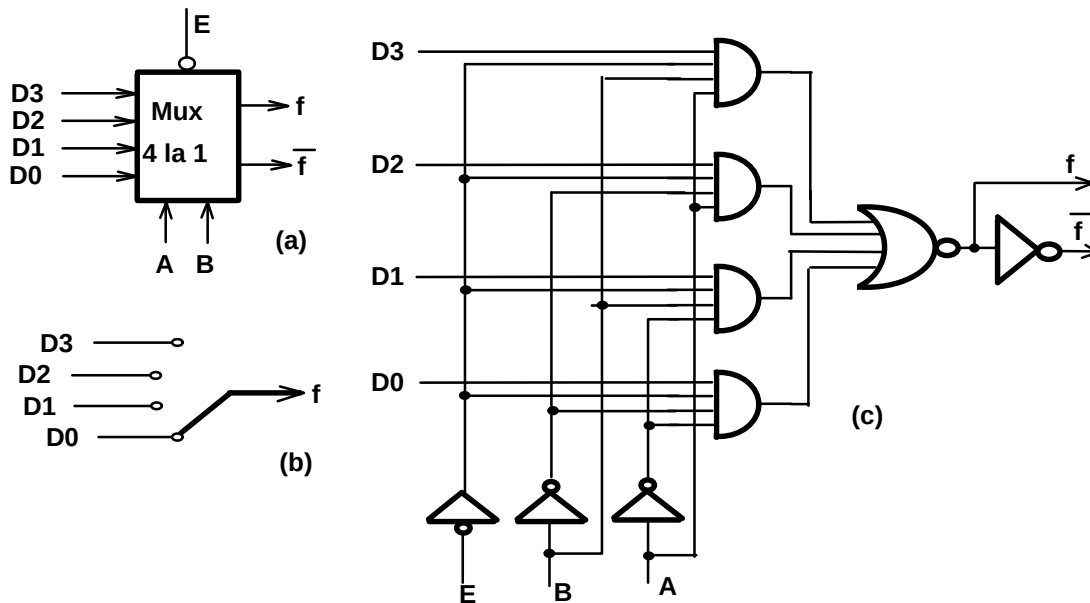


Fig. 2.20. Simbolul, schema functionala si schema logica a unui multiplexor 4 la 1.

2.2.9 Demultiplexoare si decodificatoare

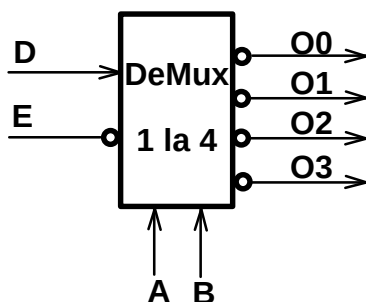


Fig. 2.21. Demultiplexor 1 la 4

Un demultiplexor indeplineste functia opusa unui multiplexor. El poate fi utilizat la convertirea unui sir de semnale numerice transmise serial, într-o forma paralela. Diagrama bloc a acestui circuit este prezentata in figura 2.21. Linia de intrare D poate fi conectata la oricare dintre cele patru linii de iesire, prin semnalele de control A si B . Fiind posibile 4 combinatii distincte ale celor doua semnale de control A si B , circuitul va avea in acest caz patru linii de iesire. Astfel, pentru $A=0, B=0$, semnalul D este transferat la iesirea $O0$ (la activarea circuitului, prin semnalul $E=0$), etc.

Un decodicator este, in esenta, un demultiplexor fara intrari de date. El convertește un cod pe n biti in maxim 2^n iesiri individuale. Figura 2.22 prezinta diagrama bloc si tabela de adevar a unui decodicator 3 la 8. Iesirile decodicatorului sunt active pe 0.

Decodificatoarele sunt utilizate în sistemele microprocesor, servind la selectia cipurilor de memorie, a porturilor sistemului, etc. De exemplu, decodicatorul din figura 2.22 poate fi utilizat la selectia unuia din 8 cipuri de memorie, în functie de codul de selectie al liniilor de intrare, furnizat de catre microprocesor. Decodicatorul are o linie de activare. De multe ori, decodificatoarele sunt dotate cu mai multe linii de activare, (unele active pe nivel logic "0", altele pe nivel logic "1"), ceea ce maresc flexibilitatea acestor circuite.

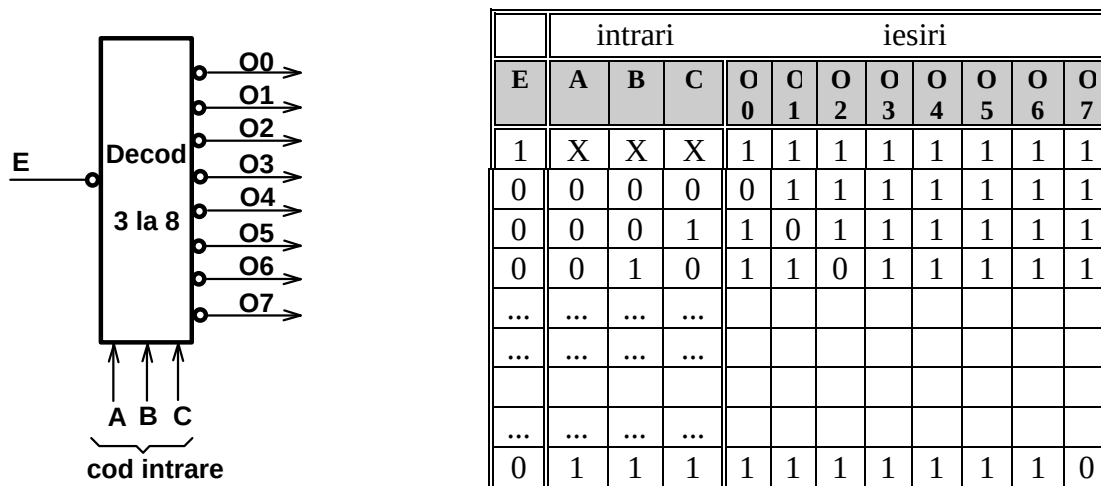


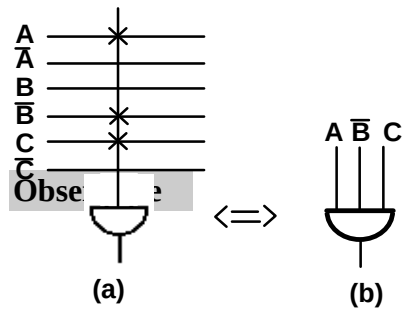
Fig.2.22. Schema și tabela de adevăr a unui decodicator 3 la 8

Dintre circuitele MSI întâlnite deasemenea în sistemele microprocesor mai menționăm encoderele (circuite complementare decodificatoarelor).

Probleme specifice sistemelor microprocesor sunt și cele legate de circuite de generare a semnalelor de paritate, comparatoare logice, circuite aritmetice, etc. Nu vom mai intra în detalii, care vor fi date, în măsura necesității, la apariția respectivelor circuite în sistemele ce vor fi prezentate, după caz.

2.2.10 Circuite logice programabile (PLD)

O categorie aparte de circuite logice, utilizate tot mai mult la sinteza unor părți componente de logică combinatională și / sau secvențială, din cadrul sistemelor digitale, o reprezintă **circuitele logice programabile (PLD - programmable logic devices)**. În esență, aceste circuite conțin combinații de porți logice "SI" (AND) și "SAU" (OR), *inversoare* (NOT), circuite *tri-state*, și *bistabile*. Ele pot fi programate pentru a implementa funcții logice specificate. Uzual, circuitele PLD sunt circuite cu o singură utilizare (se pot programa o singură dată); există mai nou și circuite PLD *reprogramabile*, utile în fazele de proiectare și testare a schemelor implementate. Gama de tipuri de asemenea circuite este destul de largă; uzual, sunt fabricate în tehnologie CMOS bipolar.



În esență, circuitele *PLD* implementează expresii booleene utilizând matrici (rețele) de porți *AND* pe linii, conectate la porți *OR* pe coloane, ieșirile *PLD*-ului fiind ieșirile portilor *OR*.

Vom prezenta în continuare principalele tipuri de circuite *PLD* și caracteristicile lor de bază.

Din motive de simplitate a schemelor, se utilizează simbolic reprezentarea din figura 2.23.a, echivalentă cu schema din figura 2.23.b, pentru a indica conectarea unor semnale la intrările unei porți (*AND* sau *OR*). Semnele (x), de la intersecția unei linii cu o coloană, indică prezența unui fuzibil intact, care conectează semnalul de intrare la poarta logică *AND* (sau *OR*) de pe coloana respectivă. Inițial, toate semnalele de intrare sunt conectate la toate portile circuitului (matricea are toate fuzibilele intacte). Prin programarea circuitului, se determină de fapt distrugerea unora dintre fuzibile, astfel încât în final, prin conexiunile care rămân intacte, să se sintetizeze funcția logică dorită între semnalul de intrare și cel de ieșire. Fiecare combinație a semnalelor de intrare care se aplică la intrarea unei porți *AND* (sau *OR*) a circuitului *PLD*, poartă numele de "**product-term**", sau "**P-term**" (v.fig. 2.23, coloana conectând semnalele A, B̄, C la intrarea porții *AND*).

Fig. 2.23 Simboluri utilizate pentru descrierea circuitelor *PLD*.

2.2.10.1 Circuite *PAL* - Programmable Logic Array

Sunt circuite care conțin o matrice programabilă de porți *AND*, conectate la o structură fixă cu porți *OR* (v.fig.2.24). Între semnalele de intrare (*A*, *B*, *C*, *D*, *E* și *F* în exemplul din figura 2.24), se implementează pe baza matricii de conexiuni, combinațiile necesare pentru a sintetiza funcția logică dorită și a obține semnalul de ieșire ($\bar{X}$ în exemplul din figura 2.24). După cum se remarcă din figura 2.24, uzual sunt selectabile în matrice semnalele de intrare sau negatele lor. De asemenea, un "*P-term*" poate fi tri-statat și se poate aplica sub formă de semnal de intrare în circuit. În figura 2.24 funcția booleană implementată are expresia (dedusă ușor din analiza schemei de conexiuni a semnalelor circuitului):

$$X = p + q + r + s = ABCD + \bar{A}\bar{B}CD + E\bar{F}$$

Circuitele *PAL* pot fi prevăzute la ieșire și cu registre, utilizabile pentru a sintetiza circuite secvențiale.

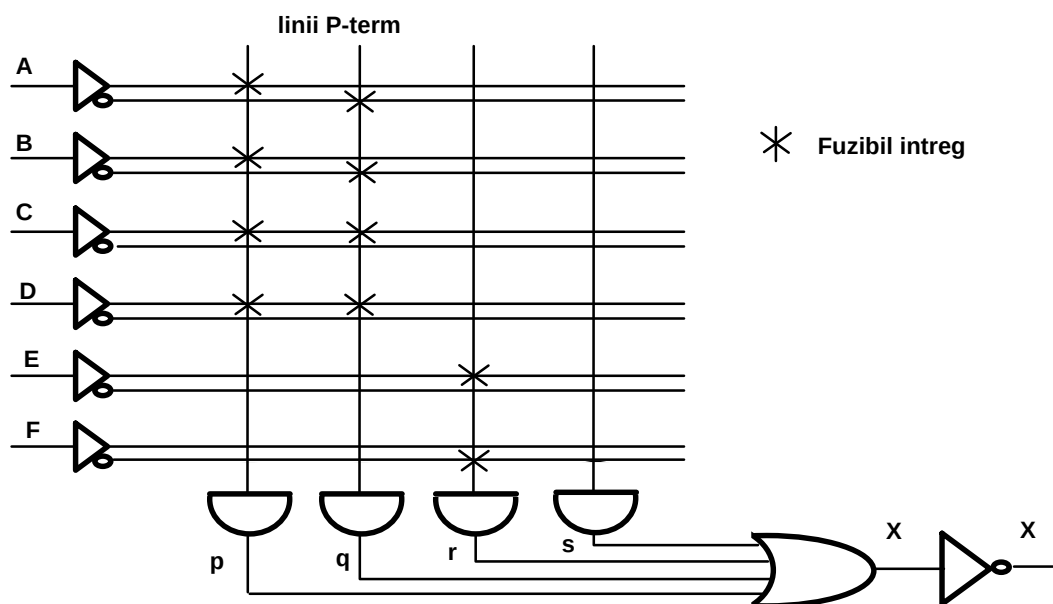


Fig.2.24. Structura de principiu a unui circuit PAL

2.2.10.2 Circuite PROM - Programmable Read Only Memory

Uzual circuitele **PROM** sunt incadrate si utilizate functional ca memorii cu continut nealterabil (fix), stocand coduri de program, seturi de tabele de date constante, etc. Totusi, este posibila si uneori convenabila utilizarea lor si in proiectarea circuitelor logice. Astfel, structura interna a unui circuit **PROM** este o retea fixa de porti **AND** si o retea programabila de porti **OR**. Figura 2.25 prezinta structura unui **PROM** cu trei semnale de intrare (adrese) - A, B, C, si trei semnale de iesire (practic, este un circuit de memorie cu 8 celule de cate 3 biti fiecare). Pentru *fiecare combinatie* a semnalelor de intrare (adresa), se selecteaza in mod unic o combinatie a semnalelor P0 la P7 (reprezentand 8 *P-term* fiksi). Corespunzator acestuia, prin programarea retelei OR, se genereaza semnalele O/P0, O/P1 si O/P2. (De remarcat ca fiecare semnal de iesire poate reprezenta o combinatie a mai multor *P-term*, ceea ce nu se intampla la circuitele **PAL**). Pentru exemplul din figura 2.25, semnalele de iesire au expresiile:

$$\begin{aligned} O/P0 &= P0 + P2 = \overline{CBA} + \overline{CBA} \\ O/P1 &= P2 + P4 = \overline{CBA} + CBA \\ O/P2 &= P3 + P4 + P5 = \overline{CBA} + \overline{CBA} + \overline{CBA} \end{aligned}$$

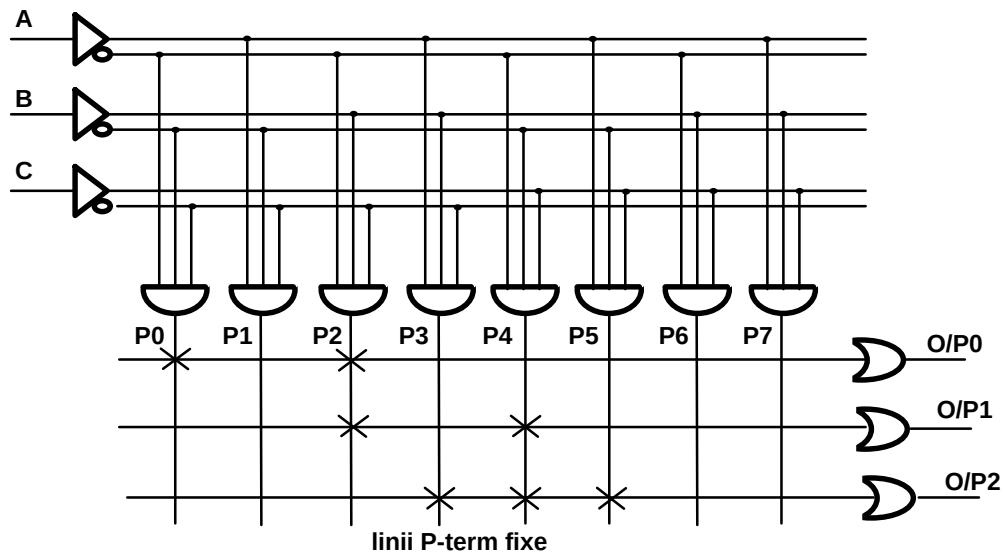


Fig.2.25. Schema de principiu a unui circuit PROM utilizat pentru sinteza functiilor logice

2.2.10.3 Circuitele FPLA - Field Programable Logic Array.

Circuitele **FPLA** reprezinta primele tipuri de circuite *PLD*. Contin retele programabile atat pentru portile *AND* cat si pentru cele *OR*. La fel ca si circuitele *PAL*, pot contine si circuite tri-state pentru conectarea iesirilor la intrari, cat si registre pentru implementarea circuitelor secventiale. Circuitele *FPLA* sunt mai flexibile decat circuitele *PAL* sau *PROM*, dar mai lente decat acestea. Mai mult, deoarece se prefera uzual proiectarea cu retele *AND* sau cu retele *OR* (deci utilizand doar unul dintre cele doua tipuri), ele nu sunt neaparat utilizate la capacitatea lor maxima. Figura 2.26 prezinta structura unui circuit *FPLA*, sintetizand semnalele:

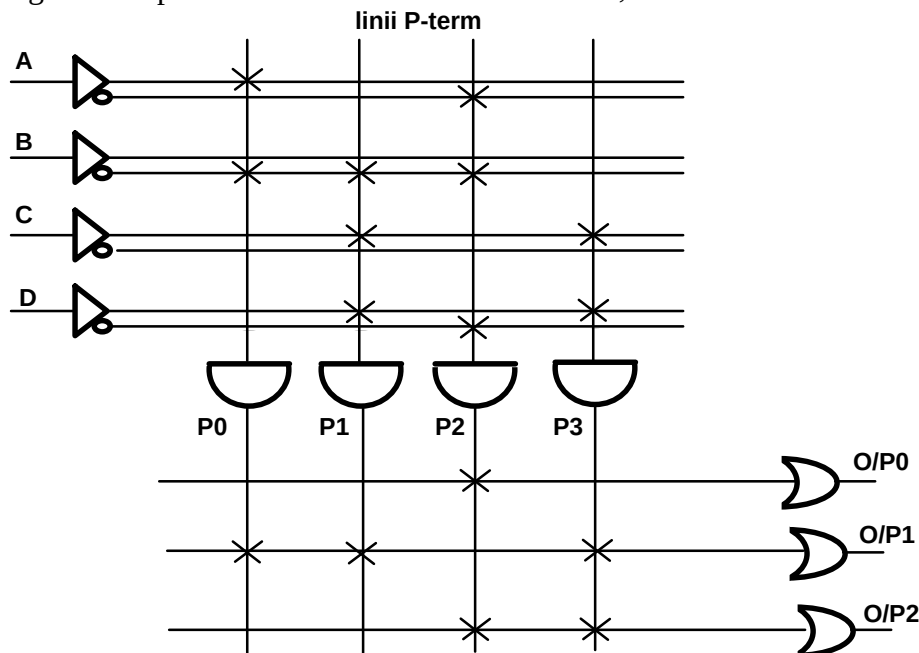


Fig.2.26. Schema de principiu a unui circuit FPLA

$$\begin{aligned}
 P0 &= A \cdot \bar{B}, & P1 &= \bar{B} \cdot C \cdot D, & P2 &= \bar{A} \cdot \bar{B} \cdot \bar{D}, & P3 &= C \cdot D \\
 O/P0 &= P2, & O/P1 &= P0 + P1 + P3, & O/P2 &= P2 + P3
 \end{aligned}$$

2.2.10.4 Circuitele EPLD - Erasable PLD.

Circuitele *PLD* din prima generatie au utilizat tehnologia bipolară, cu fuzibilele din rețeaua programabilă care odată distruse nu mai puteau fi refacute. Acum, există circuite *PLD* realizate în tehnologia circuitelor *EPROM*, deci *reprogramabile*. Astfel, ele pot fi programate, testate și, dacă este necesar, reinitializate (prin expunere la radiații ultraviolete), și reprogramate. Aceste circuite sunt foarte utile în etapa de dezvoltare a unui produs nou, când este uzuală modificarea, iterativă, a structurii circuitului, până la implementarea soluției finale. Evident, prețul acestor circuite este sensibil mai ridicat decât la circuitele cu o singură utilizare (sunt realizate în capsula ceramică, nu de plastic, sunt prevăzute cu o fereastră de cuarț, pentru ștergerea informației, prin radiații ultraviolete).

Mai trebuie specificat că unele circuite *PLD* evoluat sunt prevăzute la ieșire cu structuri complexe, numite "**macrocelule**" (v.fig.2.27), care înlocuiesc circuitele *OR* și bistabilele, cu structuri flexibile, ce pot fi programate în diverse configurații. Astfel, ieșirile pot fi programate să fie active high sau low, se pot utiliza semnalele de ieșire direct sau ca semnale de intrare, intrările asincrone ale registrelor pot fi programate, și chiar tipul registrului poate fi setat (*D*, *SR*, *T* sau *JK*).

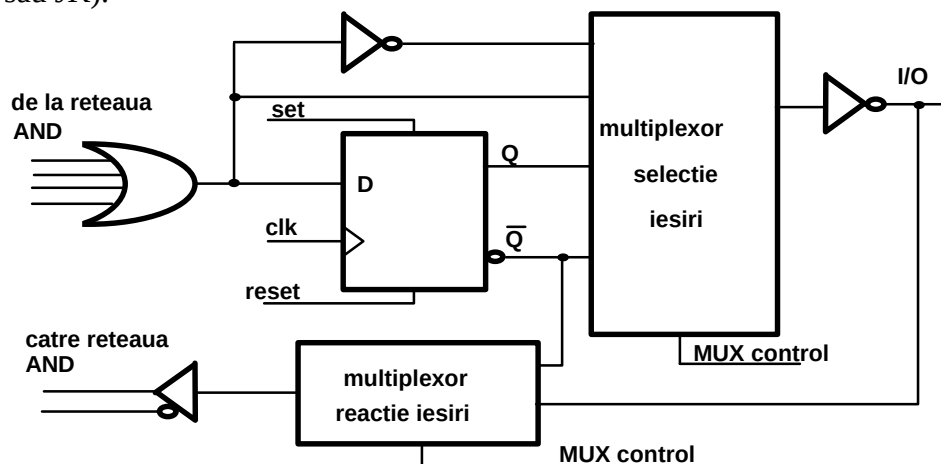


Fig.2.27. Structura de principiu a unei "macrocelule" a unui circuit PLD evoluat

2.2.10.5 Circuite PLS - Programable Logic Sequencer

Acestea reprezintă circuite *PLD* optimizate pentru utilizarea în sinteza circuitelor secvențiale complexe (**masini de stare - state machines**). Au o structură de bază similară cu cea a circuitelor *FPLA* (deci rețele programabile *AND* și *OR*), și bistabile încorporate pentru memorarea variabilelor de stare. Ca un exemplu, circuitul *PLS167* are 6 bistabile *SR*, 48 *P-term*, și o rețea *OR* programabilă.

Pentru sinteza funcțiilor logice și programarea circuitelor *PLD*, se utilizează diverse **programe și limbaje specializate**, uzual specifice pentru anumite familii de circuite *PLD*. Acestea

faciliteaza munca proiectantului de circuite, transpunand proiectarea de la sinteza schemelor logice propriu-zise, la elaborarea functiilor (expresiilor) logice ce se vor implementa. Pe baza descrierii facute de catre utilizator, programul genereaza comenzile de programare a circuitului *PLD*, ce implementeaza structura descrisa functional de catre utilizator. Tehnici speciale si programele aferente sunt deasemenea disponibile pentru a se converti scheme existente, la care este uneori dificila sinteza functiei logice aferente, in scheme cu functionalitate similara, implementate cu circuite *PLD*.

In concluzie, reprezentand instrumente moderne de lucru, circuitele *PLD* capata o tot mai mare raspandire in proiectarea si implementarea sistemelor digitale. Proiectarea la nivel logic, posibilitatile de simulare, testare, flexibilitatea (la circuitele reprogramabile), compactitatea solutiei finale, sunt caracteristici deosebit de atractive pentru proiectantul acestor sisteme. In plus si din punct de vedere economic, aceste circuite se dovedesc de cele mai multe ori o alternativa viabila la structurile logice sintetizate cu componente logice discrete "clasice". Daca strategia abordata initial in proiectarea circuitelor logice era de a se minimiza la maximum schemele implementate, se poate arata ca la ora actuala, la preturile curente ale componentelor existente, este mai importanta minimizarea numarului global de circuite integrate folosite. Se poate ajunge ca, de exemplu, o solutie continand doar cateva circuite *MSI* complexe, chiar daca in final foloseste un procent relativ scazut din capacitatea circuitelor respective, sa fie mult mai ieftina decat o solutie optimizata la maximum, dar implementata cu circuite logice conventionale (porti, bistabile, etc.)

3. MICROPROCESOARE - CONCEPTE GENERALE

Microprocesorul reprezinta probabil cea mai importanta realizare a industriei electronice a ultimelor doua decenii. Aparitia sa este legata de necesitatea producerii unor **circuite integrate pe scara larga (LSI) universale**, apte de a fi utilizate intr-o gama extinsa de aplicatii. Pana la crearea microprocesorului, circuitele *LSI* erau proiectate pentru o aplicatie anume, ceea ce limita foarte mult extinderea masiva a productiei acestora. Era necesar un nou concept in ceea ce priveste functionalitatea unui asemenea circuit, flexibilitatea utilizarii lui. Aceasta varietate de functii si posibilitati, permitand realizarea, cu un numar redus si standardizat de circuite integrate, a diverse sisteme automate complexe, a fost implementata cu ajutorul sistemelor microprocesor. Legat, prin interfete specifice, de lumea inconjuratoare, un astfel de sistem are posibilitatea culegerii de informatii din exterior (marimi numerice, analogice, semnalizari), a prelucrarii lor (diversi algoritmi de calcul si decizie), si a actionarii spre exterior (dand comenzi numerice, analogice, semnalizari diverse, comunicand rezultatele operatiilor efectuate, etc.).

Preluand, uneori chiar dezvoltand structura functionala de baza a oricarui calculator electronic "clasic", aceasta gama deosebit de larga de sisteme, denumite **microcalculatoare**, poate implementa aplicatii limitate doar de marginile imaginatiei omenesti. Aria lor de utilizare este considerabil marita fata de cea a calculatoarelor electronice mari, prin dimensiuni mult reduse, functionare nepretentioasa din punct de vedere al conditiilor de mediu ambiant, fiabilitate ridicata, viteza de lucru comparabila (tot mai mare la tipurile noi de microprocesoare), si, nu in ultimul rand, pretul, in scadere exponentiala, al acestor tipuri de echipamente.

Utilizatorii actuali, cat si cei potentiali, ai acestor puternice instrumente, trebuie sa stapaneasca intr-o oarecare masura notiunile fundamentale, atat ale structurii constructive (hardware) a microcalculatoarelor, cat si a celei referitoare la programarea lor (software). Pe de o parte, cel ce implementeaza structura unui astfel de sistem trebuie sa cunoasca disponibilitatile de programare (setul de instructiuni) ale microprocesorului utilizat, iar cel ce utilizeaza sistemul are nevoie de informatii constructive si functionale pentru a implementa in mod optim, ca performante, o aplicatie data (prin realizarea unui program adecvat).

Din aceste motive, desi prezentarea este facuta in general distinct, nu s-a realizat o separare neta la prezentarea structurii hardware sau software a microcontrolerelor.

Pentru punerea in acord a cunostintelor de baza, necesare la intelegerea principiilor de functionare ale unui microprocesor, ale programarii acestuia, aceasta prezentare face o trecere in revista a notiunilor fundamentale legate de sistemele microprocesor.

Se considera ca cititorul poseda un bagaj minim de notiuni legate de structura si functionarea circuitelor logice (combinationale si secventiale), cat si de aritmetica unui calculator (baze de numarare, operatii binare etc.).

3.1 MICROPROCESORUL --- PREZENTARE FUNCTIONALA

Intrebarile la care vom incerca sa raspundem in urmatoarele paragrafe sunt: ce este un microprocesor? ce poate face el? cum functioneaza? cum interactioneaza cu lumea inconjuratoare? cum se implementeaza, pe baza lui, un sistem microprocesor?

Cunoasterea raspunsurilor la aceste intrebari ofera baza necesara pentru evaluarea justa a posibilitatilor si performantelor diverselor sisteme, in vederea alegerii solutiei optime (combinatia hardware - software cea mai indicata) pentru o aplicatie data.

Deci, ce este un microprocesor? In esenta, un circuit integrat *LSI*, reprezentand elementul central (**CPU**¹), ca importanta si posibilitati, in efectuarea sau coordonarea tuturor operatiilor ce au loc in functionarea unui microcalculator.

Intr-un mod oarecum figurat, el reprezinta un circuit logic de o extrema complexitate, reprezentabil, din punct de vedere functional, printr-o structura formata din elemente de logica combinationala si secventiala. Elementul specific, care diferentiaza microprocesorul de o astfel de configuratie clasica, si care ii confera totodata exceptionala gama de posibilitati de utilizare, practic intr-un numar nelimitat de aplicatii, il reprezinta flexibilitatea sa, la nivelul *software* al sistemului. Aceasta face ca, de fapt, implementarea acelei structuri logice echivalente, pentru o aplicatie data, sa nu reprezinte o interventie la nivelul structurii constructive, *hardware*, a sistemului microprocesor, ci sa se obtina prin implementarea unui program specific. Modificarea aplicatiei nu reprezinta altceva decat modificarea programului microprocesorului.

Astfel, in esenta, se implementeaza structuri logice prin program, deci prin *software*. Totusi, trebuie evidentiata faptul ca dezvoltarea de noi programe consuma un timp ce poate fi apreciabil, si uneori modificarile nu sunt asa de simple pe cat par a fi. Proiectarea cu microprocesoare implica trecerea de la proiectarea cu circuite logice combinationale si secventiale, la combinatia hardware-software optima oferind performante maxime pentru un cost al instalatiei dat. Proiectantul sistemului microprocesor trebuie sa stapaneasca atat proiectarea circuitelor logice, cat si tehnica programarii. Calitatile cerute unui bun programator sunt insa similare, in esenta, celor cerute proiectantului de structuri logice. Ceea ce trebuie remarcat de la bun inceput este faptul ca microprocesorul are posibilitatea de a executa actiuni proprii unui anumit numar de instructiuni (uzual cateva zeci), care alcatuiesc setul sau de instructiuni (si care este specific fiecarui tip de procesor in parte). Pe baza acestor instructiuni se elaboreaza programul (de catre programator) pe care il va executa microprocesorul. Memorat intr-o memorie, programul este "citit" (adus din memorie in microprocesor), instructiune cu instructiune, care, odata decodificate, determina actiunea

¹ CPU: Central Processing Unit = unitate centrala de prelucrare

dorita. O modificare in comportarea sistemului se obtine asadar printr-o modificare (sau mai multe) in program. Evident, programul trebuie sa contina doar instructiuni al caror cod este "inteles" de catre microprocesor (apartin setului lui de instructiuni). Orice eroare de program poate determina functionarea partial sau complet defectuoasa a sistemului. (Aceasta va determina si particularitatea depanarii unui sistem microprocesor. Este necesara o depanare *hardware* a schemei de conexiuni, a bunei functionari a circuitelor, cat si una *software*, a programului).

Nu este locul si nici nu este necesar, in vederea utilizarii adecvate a unui microprocesor, sa se abordeze structura de detaliu, constructiv interna, a acestuia. Punctele de vedere din care se va face prezentarea sa vor fi cel **functional**, legat de actiunile specifice efectuate de catre microprocesor, si cel **temporal-secvential**, legat de succesiunea si durata acestora. Legatura intre aceste doua abordari este extrem de importanta.

3.1.1 Functiile de baza ale microprocesorului

La ora actuala, exista disponibil un numar foarte mare de microprocesoare, produse de catre diferite firme. Desi exista multe deosebiri in structura si tehnologia lor de fabricatie, toate microprocesoarele sunt proiectate pentru a efectua un numar de operatii de baza. Dupa cum s-a specificat, de altfel, aceste caracteristici functionale comune provin din filozofia crearii microprocesoarelor, ca instrumente complexe utilizate la implementarea unor sisteme numerice flexibile, puternice, si la un pret deosebit de avantajos.

Structura de baza a unui sistem microcalculator este prezentata in figura 3.1. Se remarca faptul ca intr-o sectiune de memorie sunt memorate atat secventele de instructiuni ce alcatuiesc programul, cat si datele numerice (de intrare, sau rezultate intermediare sau finale). Prin sectiuni specifice de intrare sau iesire, calculatorul comunica cu perifericele exterioare, care pot fi atat echipamente standard (tastatura, afisaj, etc.), cat si interfete specifice de conectare la un proces fizic controlat (traductoare, elemente de executie, etc.).

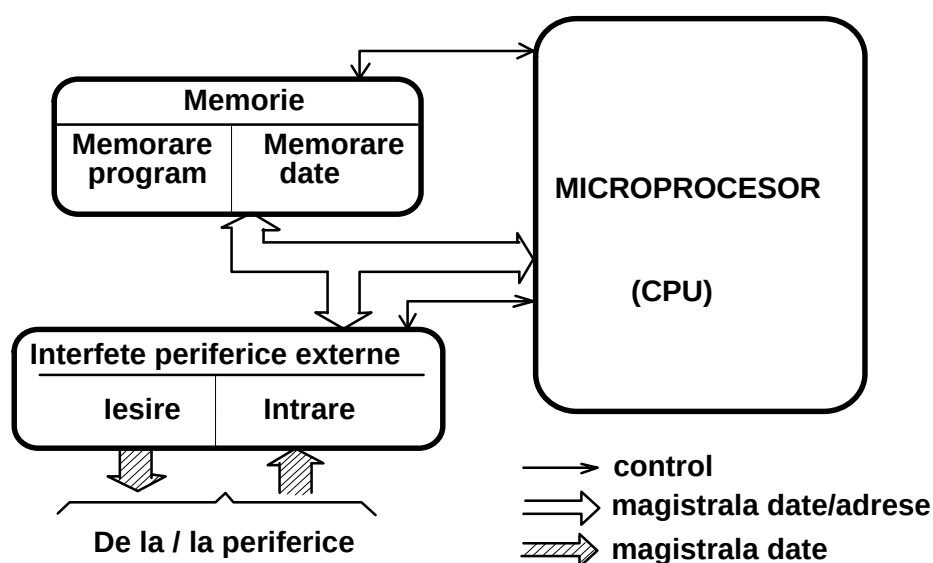


Fig.3.1 Structura de baza a unui sistem microcalculator

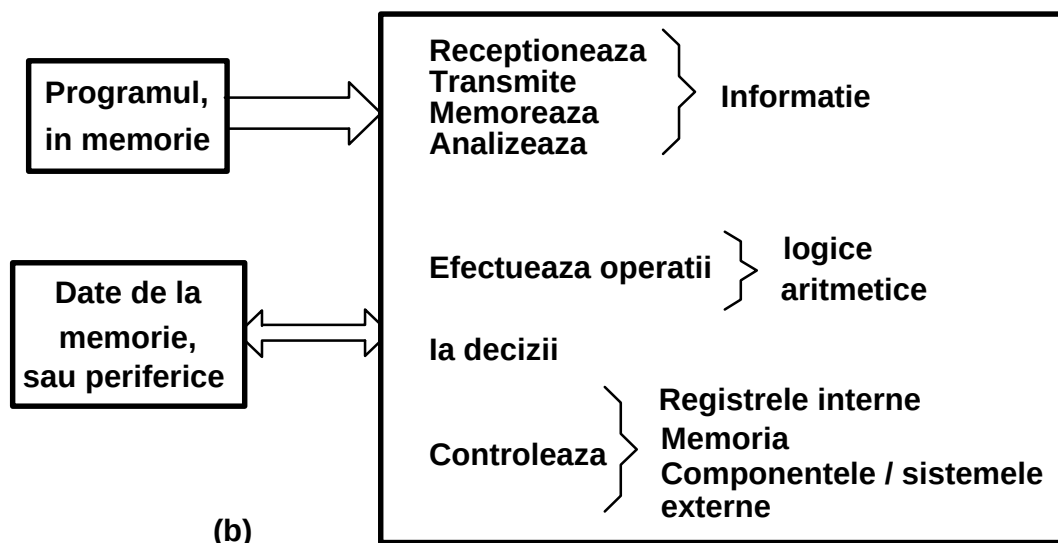


Fig.3.2. Reprezentarea functiilor unui microprocesor, corelat cu memoria si perifericele sistemului

Figura 3.2 prezinta structura functionala a unui microprocesor in contextul utilizarii memoriei si a perifericelor sistemului. Se disting pe grupe functionale: manipularea informatiilor (instructiuni pentru microprocesor, cat si date transmise sau primite de la diferite componente ale sistemului), executia diferitelor operatii de calcul, si controlul si supervizarea tuturor componentelor din sistem, si a operatiilor efectuate in diferitele sectiuni ale acestuia.

Functional, un sistem microprocesor (ca de altfel orice calculator numeric), prezinta cinci functii de baza: functia de intrare (**INPUT**), care permite legatura intre lumea exterioara si sistem; functia de iesire (**OUTPUT**), care permite legatura intre sistem si lumea exterioara; functia de memorare (**MEMORY**), care permite pastrarea informatiilor (date, rezultate), si (uneori) a instructiunilor programului; functia aritmetico-logica, (implementata prin **ALU**¹), care permite efectuarea operatiilor aritmetice si logice in sistem; functia de **CONTROL**, care inglobeaza totalitatea actiunilor de secventializare si control ale activitatii sistemului.

Aceasta delimitare functionala nu este stricta, in sensul existentei intr-o proportie variabila, la diverse tipuri de microprocesoare, a unora din cele 5 sectiuni. In speta, sectiunile de intrare si iesire ale microprocesorului apar in general in mica masura la nivelul microprocesorului ca legaturi directe spre exteriorul sistemului; deasemenea, doar unele microprocesoare, (si acelea, in general, partial) acopera ele inelele necesitatilor de memorie ale sistemului. Uzual, aceasta este exterioara unitatii centrale.

Figura 3.3 prezinta fluxul de informatie intre cele cinci componente functionale in sistemul microprocesor. Uzual, informatia este adusa, prin functia de intrare, (**INPUT**), sub coordonarea functiei de control, in memoria sistemului. Informatia poate fi utilizata ori pentru a "spune" microprocesorului ce are de facut (instructiuni), ori ca data in operatii aritmetice sau

¹ ALU - Arithmetic Logic Unit = unitate aritmetico - logica (UAL)

logice. (In caz general, sub forma de instructiuni, alcatuind programul, informatia poate fi rezidenta in mod permanent in memoria sistemului).

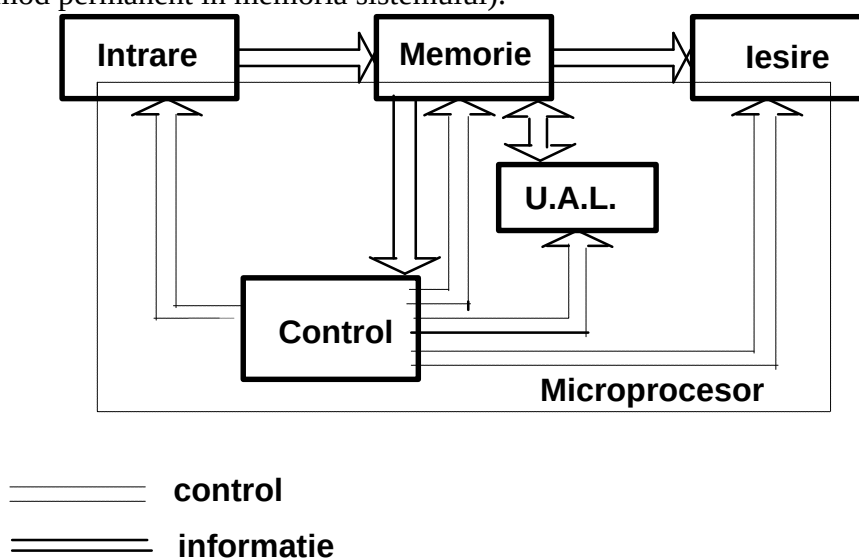


Fig.3.3. Structura functionala si fluxul de informatii dintr-un sistem microprocesor

Din memorie, tot sub incidenta functiei de *CONTROL*, informatia este citita si decodificata de catre microprocesor, executandu-se in mod secvential instructiunile programului. Datele si rezultatele sunt deasemenea vehiculate intre procesor si memorie, tot sub comanda functiei de *CONTROL*, conform programului. Dupa caz, coordonat de catre functia de *CONTROL*, se activeaza, pentru transfer de informatie spre iesirea sistemului, functia de iesire (*OUTPUT*).

Se remarca delimitarea, in figura 3.3, a functiilor sistemului, inglobate de catre microprocesor. Doua dintre sectiunile sistemului sunt continute intotdeauna integral de catre microprocesor:

Blocul ALU, care reprezinta o colectie de circuite logice complexe, programabile, capabile sa proceseze date numerice, pe baza unui set de instructiuni ale utilizatorului, uzual bazate pe trei operatii fundamentale: adunare, scadere, si operatii logice.

Sectiunea de CONTROL, care genereaza si coordoneaza toate semnalele de control necesare pentru a executa operatiile de calcul din ALU, si a sincroniza transferul datelor in sistem. O functie esentiala a sectiunii de *CONTROL* este aceea de a decodifica si executa instructiunile programului pe baza caruia functioneaza intregul sistem.

Ca o remarca speciala, modul de conectare si functionarea componentelor de baza ale unui sistem de calcul, s-au bazat si se bazeaza inca, in mod curent, in marea majoritate a structurilor utilizate, pe asa numita arhitectura de tip *von Neuman*, utilizata practic in exclusivitate la realizarea microprocesoarelor uzuale. In aceasta arhitectura, unitatea centrala de prelucrare (*CPU*), va executa cate o singura instructiune la un moment dat, intr-o secventa de program. Desi mai lenta decat in sistemele in care se executa mai multe instructiuni deodata (procesare paralela), arhitectura *von Neuman* este relativ mai simpla de implementat. Arhitecturile recentelor tipuri de microprocesoare au insa elemente de prelucrare paralela, cat si, in cazul procesoarelor *DSP*, cu asa numita arhitectura *Harvard*, o paralelizare completa a

lantului de citire / decodificare instructiuni, respectiv efectuare operatii de calcul, si transfer al datelor in sistem. Aceasta conduce, evident, la cresteri sensibile ale performantelor respectivelor sisteme, fata de cele "clasice", realizate in arhitectura *von Neuman*.

3.1.2 Aspectul temporal - secvential

Desfasurarea operatiilor efectuate de catre diversele componente functionale ale sistemului este controlata, ca durata si succesiune, de catre sectiunea de control. Aceasta sectiune selecteaza in mod corespunzator (activeaza), secvential functiunile microprocesorului, pe durate de timp fixe, determinate de specificul operatiei ce se executa. O diagrama de timp simplificata a activarii sectiunilor sistemului este prezentata in figura 3.4.

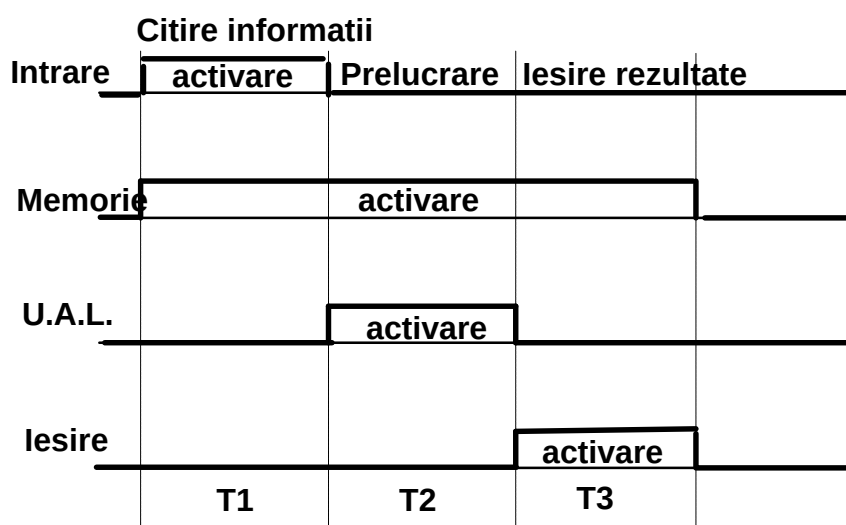


Fig.3.4. Inlantuirea temporală a operatiilor sistemului microprocesor

Pe intervalul t_1 , informatia (instructiuni si date), se preia de la intrare si se memoreaza (activandu-se, corespunzator, cele doua sectiuni, de *INPUT* si *MEMORY*). In t_2 , instructiunile din memorie sunt preluate si executate (ele alcatuind programul microprocesorului), si impreuna cu datele, prelucrate in *ALU*, determina executia dorita de operatii ale sistemului. Rezultatele sunt memorate in memorie. Intr-o ultima etapa (t_3), se dezactiveaza *ALU*, si se transfera rezultatele spre exteriorul sistemului, activandu-se functia de *OUTPUT*. Un semnal de ceas de sistem (uzual controlat cu un cristal de cuarț), este utilizat pentru sincronizarea si secventializarea diverselor operatii si semnale, pe parcursul efectuării tuturor operatiilor sistemului.

3.1.3 Reprezentarea informatiei

Implementarea electrica / electronica a functiilor calculatorului este posibila, si se realizeaza, prin utilizarea bazei 2 de numarare (numerele 0 si 1). Astfel, toate informatiile (instructiuni sau date, rezultate) se memoreaza si se prelucreaza de catre microprocesor in forma binara. Ele trebuie sa fie *interpretabile in mod unic* de catre microprocesor, prezentandu-se intr-un

format specific in acest scop. Depinzand de concepia proiectantilor microprocesorului, acest format este o *combinatie de lungime fixa, de simboluri binare*, alcatuind asa-numitele **cuvinte de instructiuni**, respectiv **cuvinte de date** (fig. 3.5). Fiecare *instructiune* din programul microprocesorului va fi formata din unul sau mai multe *cuvinte de instructiune*. Acestea "spun" procesorului ce are de facut. *Cuvantul de instructiune* consta din doua parti: **codul operatiei** (care indica ce operatie se va executa), si **operandii** (care indica asupra cui se va efectua operatia indicata de codul operatiei).

Figura 3.5 este un exemplu de reprezentare a cuvintelor de instructiune, pentru un microprocesor cu formatul cuvintelor de 8 biti. (Uzual sunt microprocesoarele cu cuvinte de **8 biti=1 byte**, in ultima vreme afirmandu-se si extinzandu-se cele de 16, 32 si chiar 64 de biti). Va exista deci cate o *combinatie unica a bitilor cuvantului de instructiune*, pentru desemnarea *codului operatiei* (combinatia 10000 in fig. 3.5.b), si a *operandului* asupra caruia se va efectua operatia dorita (combinatia 010 in fig. 3.5.b). Decodificarea cuvantului de instructiune va furniza astfel microprocesorului informatiile necesare pentru a sti ce operatie se executa, si asupra carei (caror) valori (date), se executa aceasta operatie.

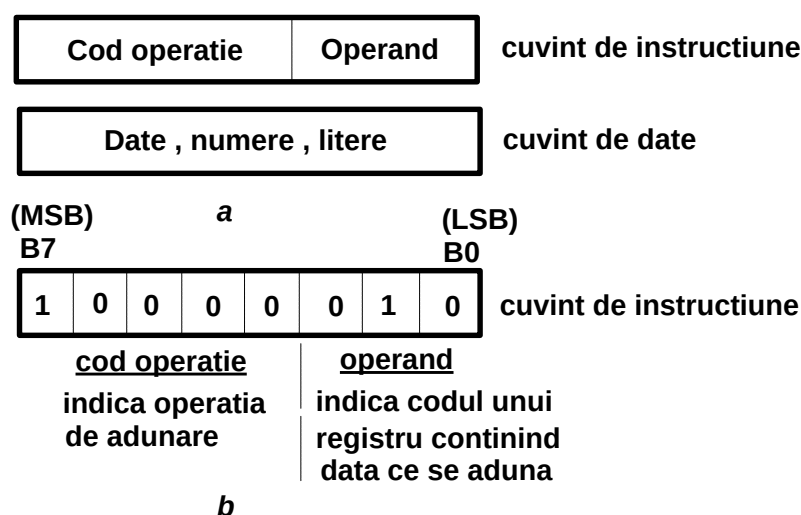


Fig.3.5. Reprezentarea informatiei (instructiuni sau date); LSB: bitul cel mai putin semnificativ (*last significant bit*); MSB: bitul cel mai semnificativ (*most significant bit*)

Instructiunile sunt *memorate in ordinea executarii* (cu exceptiile ce se vor prezenta la analiza detaliata a tipurilor de instructiuni), iar microprocesorul le *citeste si executa secvential*. In *esenta*, se citeste un cuvand de instructiune, se decodifica, obtinandu-se operandii, locatia lor si se executa operatia indicata. Apoi se trece la o noua instructiune etc.

3.1.4 Memorarea informatiei

Plecand de la o reprezentare intuitiva, de genul organizarii numerelor de telefon, fiecare cuvand de instructiune sau date are asociat, in mod unic, un numar care simbolizeaza (si se numeste), adresa sa . A citi o anumita instructiune sau data din memorie va insemna, in mod practic, a se activa, prin intermediul functiilor de control, acea celula de memorie, a carei

adresa este cea dorita. Continutul ei (instructiune sau data --- este raspunderea programatorului de a sti ce se va gasi acolo si de a se ingriji de corectitudinea amplasarii lor) este prelucrat in mod corespunzator de catre microprocesor.

Memoria sistemului este organizata sub forma unei matrice, cu liniile avand un numar de celule (biti), egal cu cel al cuvintului de instructiune sau de date, fiecărei linii corespunzându-i, in mod unic, o adresa proprie. Selectarea unei astfel de celule permite accesul de la / la microprocesor, la / de la aceasta celula de memorie, sensul transferului de informatie fiind indicat de sectiunea de *CONTROL*. Figura 3.6 prezinta modul de organizare a unei astfel de memorii, pe cuvinte de n biti (n = numarul de biti din cuvantul de date sau instructiune al microprocesorului --- 8, 16 etc.).

Aparitia unei operatii cu memoria de la o anumita adresa va determina activarea tuturor celulelor de memorie elementare de la adresa respectiva (de pe linia cu aceeași adresa). Prin intermediul semnalului $R/\overline{W}$ (citeste/scrie), sectiunea de *CONTROL* va indica operatia ce se executa, deci sensul fluxului de informatie (de la microprocesor la memorie, pentru operatie de scriere, sau de la memorie la microprocesor, pentru o operatie de citire). In mod corespunzator, informatia de la microprocesor va fi inscrisa in celulele adresate (la scriere), sau va fi preluata de la memorie de catre microprocesor (la citire). Transferul se va efectua pe toate cele n linii de date simultan (in paralel).

Dupa cum memoria contine informatii fixe (in speta programe), sau informatii alterabile in cursul desfasurarii programului, exista memorii de tip **ROM**¹, respectiv **RAM**². O memorie **ROM** poate fi doar citita; informatia din ea este permanenta si nu dispare la incetarea alimentarii calculatorului. Memoria **RAM** permite scrierea si citirea informatiilor. Ea este o memorie volatila (se pierde informatia din ea, prin oprirea alimentarii circuitului).

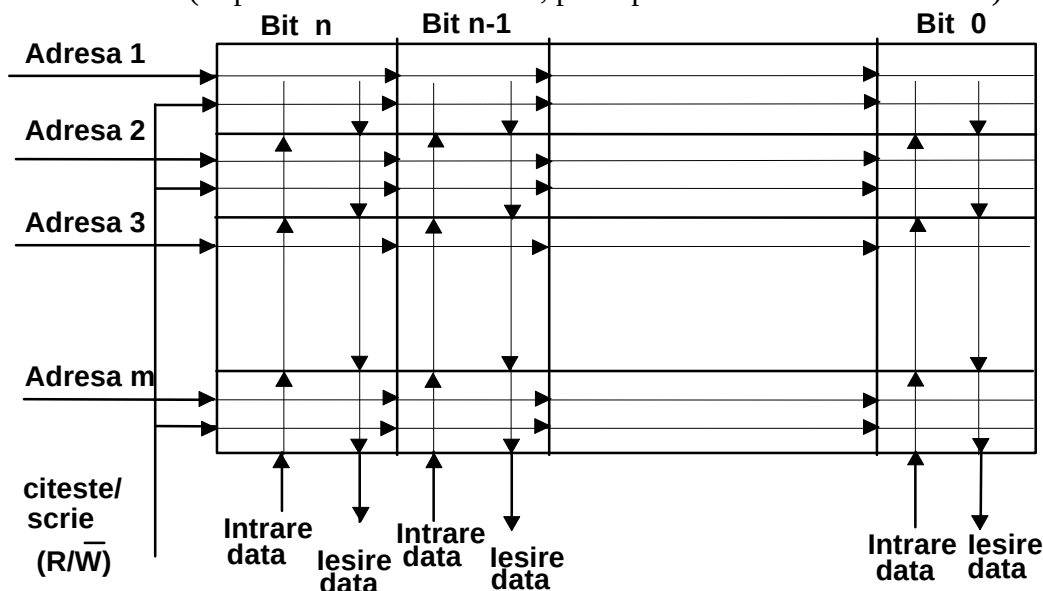


Fig.3.6. Organizarea internă a unui circuit de memorie de m cuvinte de câte n biti

¹ ROM - Read Only Memory = memorie "citeste numai"

² RAM - Random Access Memory = memorie cu acces aleator

Trebuie mentionat ca microprocesorul poate contine pe cip atat memorie *ROM* (eventual programe), cat si memorie *RAM* (de uz general, sau, sub denumirea de registre, celule cu destinatie speciala in memorarea, decodificarea si executarea instructiunilor, a operatiilor aritmetice si logice etc.).

3.1.5 Conceptul de program

Se ajunge, prin imbinarea conceptelor prezentate, de operare secventiala, reprezentare si memorare a informatiilor, la conceptul final de program. Astfel, executarea unei secvente dorite de operatii, de catre microprocesor, este obtinuta prin implementarea unei succesiuni de instructiuni, memorate in memoria sistemului, alcatuind programul acestuia. Fiecare operatie dorita a sistemului va fi implementata printr-o secventa de una sau mai multe instructiuni. Odata implementat programul si pornita executia sa , sistemul va actiona conform instructiunilor din program, realizand functiile si operatiile dorite.

3.2 CONCEPTUL DE MAGISTRALA

Sunt necesare, in functionarea unui sistem microprocesor, legaturi multiple si complexe intre diferitele parti functionale ale acestuia. **Grupurile de interconexiuni cu functii similare, legand diversele sectiuni ale unui sistem microprocesor, se numesc magistrale**. Ele conecteaza *semnalele de date, adrese, control*, ale unei sectiuni, cu semnalele omologe ale celorlalte sectiuni, constituind calea de legatura dintre ele.

Exista doua seturi de magistrale in orice sistem microprocesor: ***magistrale interne*** (reprezentand canalele de legatura intre diversele unitati functionale din unitatea centrala (*CPU*) a sistemului) si ***magistrale externe*** (reprezentand caile de comunicatie intre *CPU* si componentele externe acestuia). Fiecare grup de magistrale poate fi subdivizat in trei categorii, dupa tipul informatiei transferate: ***magistrale de adrese, de date, si de control***. O magistrala poate avea mai multe linii, permitand transferul simultan al unui cuvint de informatie (adresa sau data), caz in care magistrala se numeste magistrala paralela. Exista deasemenea magistrale seriale, pentru care datele sunt transmise multiplexat in timp (un cuvint este transmis bit cu bit, pe aceeasi linie), care sunt, in mod evident, mult mai lente decat magistralele paralele. Practic, toate magistralele interne ale *CPU*, cat si cele externe care conecteaza *CPU* cu memoria, si cu majoritatea interfetelor de intrare / iesire ale sistemului, sunt organizate ca magistrale paralele. Legaturi seriale se utilizeaza mai ales pentru transmiterea datelor la / de la echipamente aflate la distanta de *CPU* (terminale, linii telefonice, etc.).

In figura 3.7 se reprezinta o structura de principiu a modului de interconectare intre diferitele elemente componente ale sistemului microprocesor prin intermediul diferitelor magistrale ale acestuia. Deoarece unitatea centrala nu poate alimenta simultan un numar prea mare de circuite exterioare (exista valori limitate ale curentilor pe care acesta ii suporta sau genereaza), este necesara amplificarea semnalelor de pe magistralele sistemului. Aceasta adaptare a

semnalelor este efectuată cu diferite tipuri de circuite, distingându-se în esență următoarele tipuri: *circuite tampon amplificatoare de magistrală (drivere)*; *circuite tampon bidirectionale (buffer bidirectionale)*; *circuite pentru reținerea datelor (latch)*.

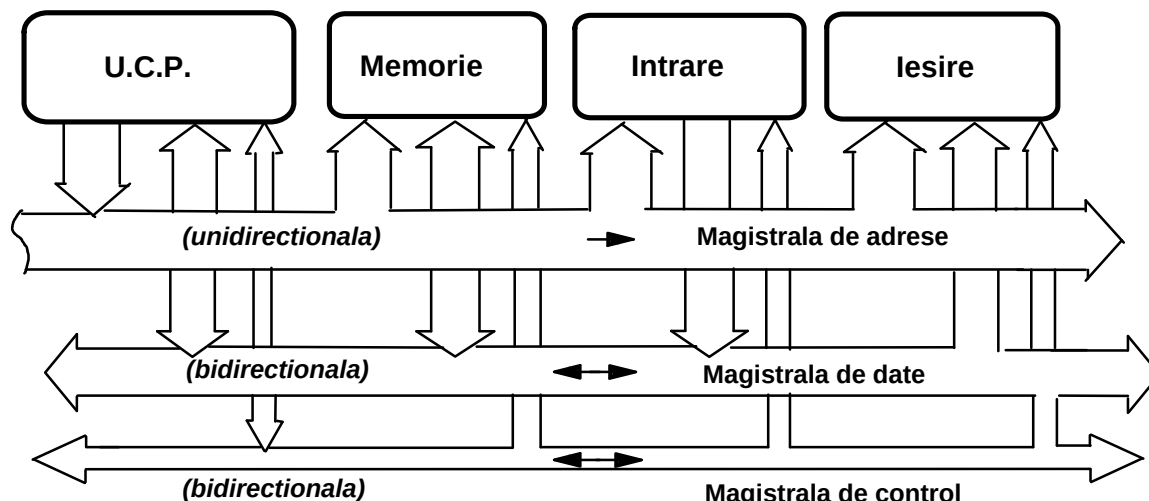


Fig.3.7. Structura unui sistem microprocesor. Magistralele sistemului.

Figura 3.8 prezintă utilizarea *circuitelor tampon (uni și bidirectionale)*, pe magistralele sistemului. Semnalele obținute la ieșirea acestor circuite formează magistrale similare celor din care provin. Uzual, se utilizează circuite tampon unidirecționale pentru semnalele de pe magistralele de adrese și de control, și circuite tampon bidirectionale, pentru magistralele de date (v. par. următoare). În cazurile în care microprocesorul se interfațează cu periferice mult mai lente decât unitatea centrală, este necesar ca informația să fie prezentă la intrarea acestor circuite, până la preluarea ei de către respectivul circuit. În aceste cazuri, se pot utiliza circuite pentru reținerea datelor (latch - reprezentând în esență, registre paralele), în care informația prezentă pe pinii de intrare, de pe o magistrală a sistemului, este înscrisă la primirea unui semnal de control (de scriere), generat direct de către microprocesor sau de către un circuit decodificator. Figura 3.9 prezintă structura unui asemenea montaj, care menține la ieșirea către un afisaj cu segmente informația dorită, până la înscriserea altor date, la un acces ulterior al circuitului latch din schema.

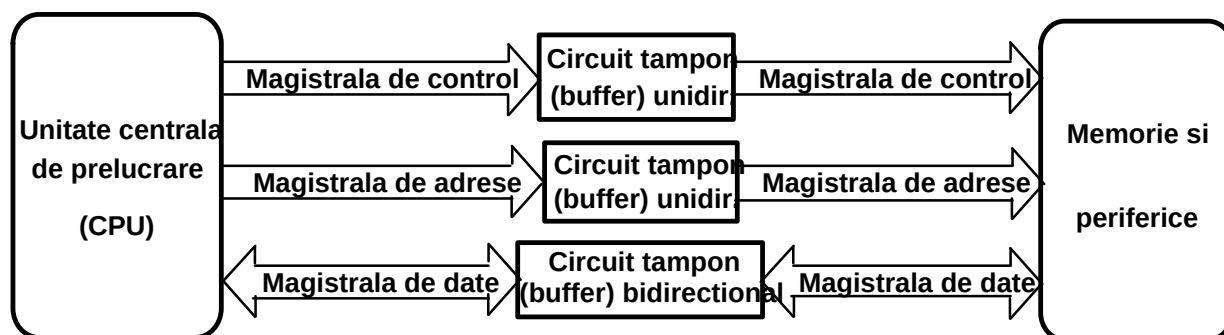


Fig.3.8. Utilizarea circuitelor tampon (buffer)

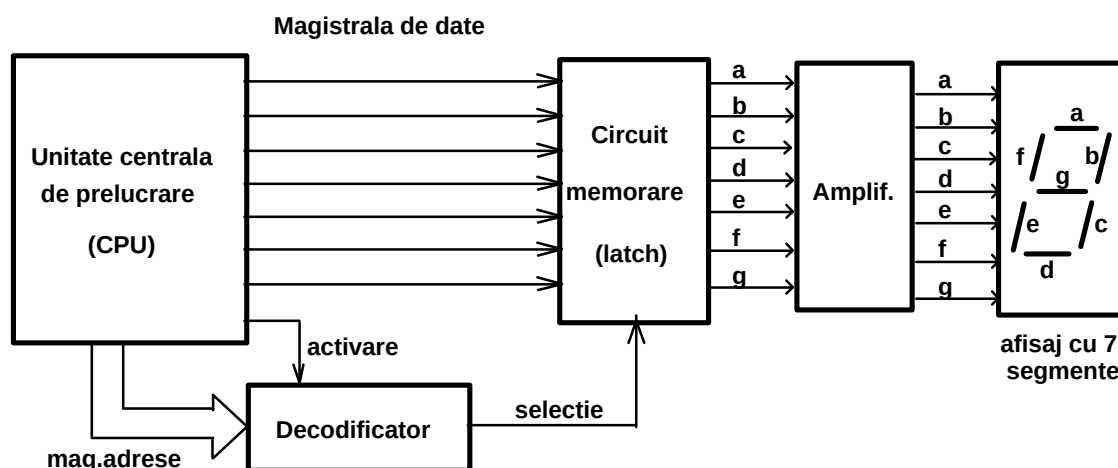


Fig.3.9. Utilizarea circuitelor de tip latch

3.2.1 Magistrala de date

Magistrala de date este destinata atat transferului *unidirectional* de instructiuni de la memorie (citire a programului), cat si celui *bidirectional*, de date intre memorie, unitatea centrala si / sau interfetele de intrari/iesiri (informatia parcurge magistrala in ambele sensuri, sau de la procesor la una dintre unitatile externe acestei unitati, sau de la una dintre aceste unitati spre procesor). Directia transferului de informatie este supervizata de catre sectiunea de *CONTROL* a microprocesorului, prin generarea de semnale specifice (citire sau scriere). Lungimea cuvintului microprocesorului determina numarul de linii de conexiune din magistrala de date (8, 16, 32, etc.).

Observatie Daca in structurile de tip *von Neuman*, exista o singura magistrala de date, pe care se vehiculeaza, la momente de timp diferite, atat cuvintele de program (coduri de instructiune), citite din memoria program, catre microprocesor, cat si cuvintele de date, intre memorie sau periferice si microprocesor, la executia instructiunilor, in structurile de tip *Harvard* (specifice procesoarelor *DSP*), exista magistrale independente pentru transferul cuvintelor de instructiune, respectiv a datelor propriu-zise ale programului. Astfel, este marita eficienta globala a sistemului, in acest al doilea caz.

3.2.2 Magistrala de adrese

Magistrala de adrese este o magistrala unidirectionala. Ea vehiculeaza codul binar reprezentand locatia (adresa) datei ce se va utiliza in cadrul operatiei ce se executa. Astfel, ea poate selecta adresa celulei de memorie de la care se citeste o noua instructiune, data, sau la care se memoreaza un rezultat. Multe microprocesoare utilizeaza magistrala de adrese si pentru configurarea adresei echipamentului de intrare-iesire, selectat la o operatie de *INPUT* sau *OUTPUT*. Este sarcina sectiunii de *CONTROL* de a delimita si indica, prin generarea de

semnale de control specifice, daca este vorba de o operatie de citire sau scriere din memorie sau intrare-iesire.

Adresabilitatea unui microprocesor este data de numarul de biti ai magistralei de adrese. Un numar uzual de 65536 (64 ko) celule de memorie vor necesita 16 linii de adresa pe aceasta magistrala. ($2^{16}=65536$). In general, se vor putea adresa 2^N celule de memorie prin intermediul a N linii de adresa.

Observatie Similar remarcii de la paragraful anterior, mentionam si aici ca structurile *von Neuman* au o singura magistrala de adrese, vehiculand la momente de timp diferite, adresa de unde se citeste cuvantul de instructiune, respectiv adresa la / de la care se efectueaza transferul datelor, la executia propriu-zisa a instructiunii. Structurile cu arhitectura *Harvard* au doua magistrale separate de adrese, pentru adresarea memoriei de program, respectiv a celei de date (ceea ce permite ca in paralel cu executia unei instructiuni sa se poate adresa si citi cuvantul de instructiune urmator).

3.2.3 Magistrala de control

Aceasta magistrala furnizeaza informatii suplimentare necesare pentru indicarea operatiei ce se efectueaza. Numarul de semnale de pe aceasta magistrala depinde de numarul de semnale de control necesare pentru microprocesorul utilizat. Tipice sunt *semnalul de ceas* al sistemului (care asigura functionarea secventiala cu o periodicitate fixa a intregului sistem microprocesor), *semnalele de citire/scriere in memorie*, *citire/scriere pentru intrare/iesire din sistem* etc.

Corelate cu magistrala de adrese, semnalele de pe magistrala de control permit selectia unica a echipamentului sau a celulei de memorie careia/de la care, prin intermediul magistralei de date, se transmite/preia informatia, conform operatiei ce se executa.

Un aspect important ce trebuie clarificat aici este cel ridicat de problema desfasurarii operatiilor in sistem, pe aceste magistrale, in mod corect, fara probleme de interferenta a informatiilor, care pe aceleasi cai sunt transferate in sensuri si spre/de la echipamente diferite. Raspunsul este dat de insasi functionarea secventiala, sub supravegherea functiei de *CONTROL*, a intregului sistem. Este asigurata activarea unica, dupa cum s-a mentionat, a celulelor de memorie sau a elementelor de intrare/iesire cu care se opereaza la un moment dat. Acestea sunt dotate cu *circuite tampon (buffere)* proprii. Conectarea pe liniile magistralelor, in sensul transferului de informatie pe magistrala, este efectuata prin circuitele tampon respective, care sunt selectate (deschise) doar in cazul adresarii circuitului respectiv. La fel, inscrierea informatiei intr-o anumita celula de memorie sau element de iesire se face prin activarea functiei de memorare doar pentru elementul respectiv. Astfel, la efectuarea unei operatii de transfer de informatii in sistem, va exista o singura sursa a semnalelor de pe fiecare magistrala, si va fi selectat un singur element in care datele se vor inscrie la operatia respectiva. Aceste selectii unice vor trebui asigurate prin proiectarea corespunzatoare a circuitelor de adresare si decodificare, dupa cum se va prezenta detaliat in paragrafele urmatoare.

Ceea ce poate fi evidentiat acum este faptul ca microprocesorul nu trebuie sa fie privit ca o colectie de circuite digitale, conectate intr-o maniera conventionala. Aceasta conectare intre circuite este in primul rand o functie a programului (*software*-ului) microprocesorului. Selectii, activari si operatii specifice ale elementelor sistemului microprocesor apar ca urmare a executiei instructiunilor programului. Se evidentiaza deci necesitatea stapanirii unei noi tehnici de "realizare" a acestor conexiuni logice functionale, aceea a *programarii* microprocesorului. Prin program se implementeaza astfel echivalentul unor structuri logice complexe, variabile in timp.

3.3 MICROPROCESORUL - PREZENTARE *HARDWARE*

Sa trecem acum la o examinare detaliata a structurii functionale a microprocesorului. Acesta va fi prezentat in maniera de abordare a utilizatorului orientat spre *hardware*-ul sistemului. Fiecare dintre cele cinci sectiuni de baza sunt identificate in mod unic. Trebuie insa mentionata prezenta unor portiuni ale functiei *MEMORY* atasate celorlalte sectiuni, cat si prezenta in mai multe locatii a elementelor de *INPUT---OUTPUT*. Aceasta s-a facut pentru o mai buna intelegere a operarii detaliata a *hardware*-ului microprocesorului, dupa cum se va vedea in continuare.

Figura 3.10 prezinta diagrama functionala detaliata a microprocesorului. Se vor discuta pe rand: fiecare sectiune functionala a acestuia, operatiile sale si legatura cu celelalte sectiuni.

3.3.1 Functia de control

Dupa cum s-a aratat, microprocesorul opereaza in mod secvential, determinat de catre un program memorat, format din instructiuni. Dar programul in sine nu este suficient pentru functionarea sistemului. (De exemplu, calculatorul trebuie sa stie de unde sa inceapa citirea din memorie a primei instructiuni din program). Trebuie sa fie activate in mod corespunzator semnalele de control, la momente adecvate de timp, pentru sectiunile de *MEMORY*, *ALU*, *INPUT/OUTPUT*, in vederea citirii, decodificarii si executarii instructiunilor in ordinea dorita si evitandu-se interferenta operatiilor.

Toate aceste cerinte sunt indeplinite de catre sectiunea de *CONTROL*. In esenta, sectiunea de *CONTROL* reprezinta "sistemul nervos" al microprocesorului, asigurand *secventializarea*, *decodificarea* si *activarea operatiilor* acestuia.

3.3.1.1 Ciclul microprocesorului

Realizarea operatiilor microprocesorului are loc in mod secvential-repetitiv. Semnalul de temporizare minima, in raport cu care se desfasoara toate celelalte temporizari sincronizate din functionarea acestuia, se numeste semnal de ceas (clock) al sistemului. Se defineste **ciclul de instructiune** al microprocesorului, care dureaza un numar variabil de perioade de ceas, in

functie de microprocesor si de instructiunea ce se executa, ca fiind secventa de operatii efectuate pentru a extrage o instructiune din memorie, a o decodifica si executa.

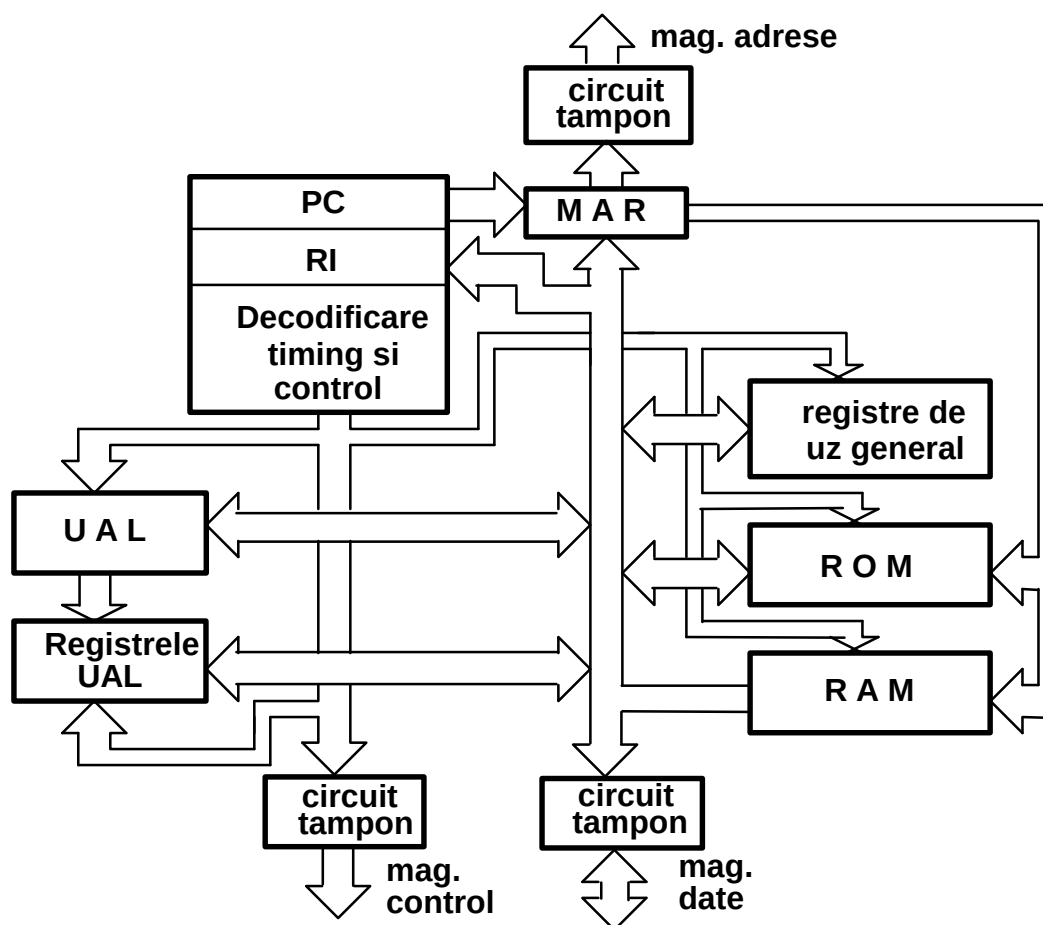


Fig.3.10. Structura internă a unui microprocesor

Ciclul microprocesorului este format din unul sau mai multe *subcicluri*, fiecare la rândul lui durând mai multe *perioade de ceas* (vezi fig. 3.11.a). Cate un subciclu este utilizat pentru obținerea fiecărui cuvânt de instrucțiune sau date, sau pentru fiecare operație de bază ce trebuie executată.

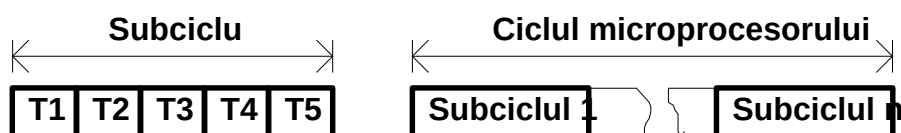


Fig.3.11.a. Structura ciclului microprocesorului

Deoarece microprocesorul execută câte o instrucțiune la un moment dat, începând cu prima instrucțiune din program, este necesar să se cunoască în permanență care instrucțiune se execută (sau va fi executată la pasul următor). Se utilizează în acest scop un registru special, numit **registru contor de program** ($PC = \text{program counter}$), care conține adresa următoarei instrucțiuni de program care va fi citită din memorie. Deoarece programul este memorat la

adrese succesive, contorul de program va fi actualizat uzual prin simpla incrementare cu 1, la fiecare transfer al unui cuvânt de instrucțiune din memorie în registrul de instrucțiuni al microprocesorului (descriș în secțiunea următoare). Astfel, putem considera registrul PC ca un **indicator (pointer)** arătând unde s-a ajuns cu execuția programului la un moment dat.

Primul subciclu al fiecărui ciclu al procesorului este utilizat pentru extragerea de informație din memorie (fetch) și are de regulă patru perioade de ceas. În prima perioadă, conținutul număratorului de adresă (PC), reprezentând adresa de unde se va citi instrucțiunea curentă, este adus pe magistrala de adrese, selectând, împreună cu semnalele de control, celula de memorie dorită. În a doua perioadă, conținutul PC este incrementat, pregătindu-l pentru următoarea extragere din memorie. În a treia perioadă, conținutul memoriei adresate este adus în microprocesor și memorat temporar. În perioadele t_4 , t_5 (prezente sau nu, în funcție de operația ce se execută), se transferă informația între registre sau se efectuează operații aritmetice simple. (În funcție de procesor și de instrucțiunea executată, un subciclu poate dura și mai multe perioade de ceas). Trebuie accentuat că întotdeauna, în primul subciclu, procesorul citește un cuvânt considerat ca primul din instrucțiune. Odată decodificat, acest cuvânt va indica ce instrucțiune se execută, și dacă este necesară citirea suplimentară de cuvinte de date din memorie (efectuată în subciclurile următoare). O instrucțiune poate fi formată din unul sau mai multe --- de regulă, până la trei --- cuvinte.

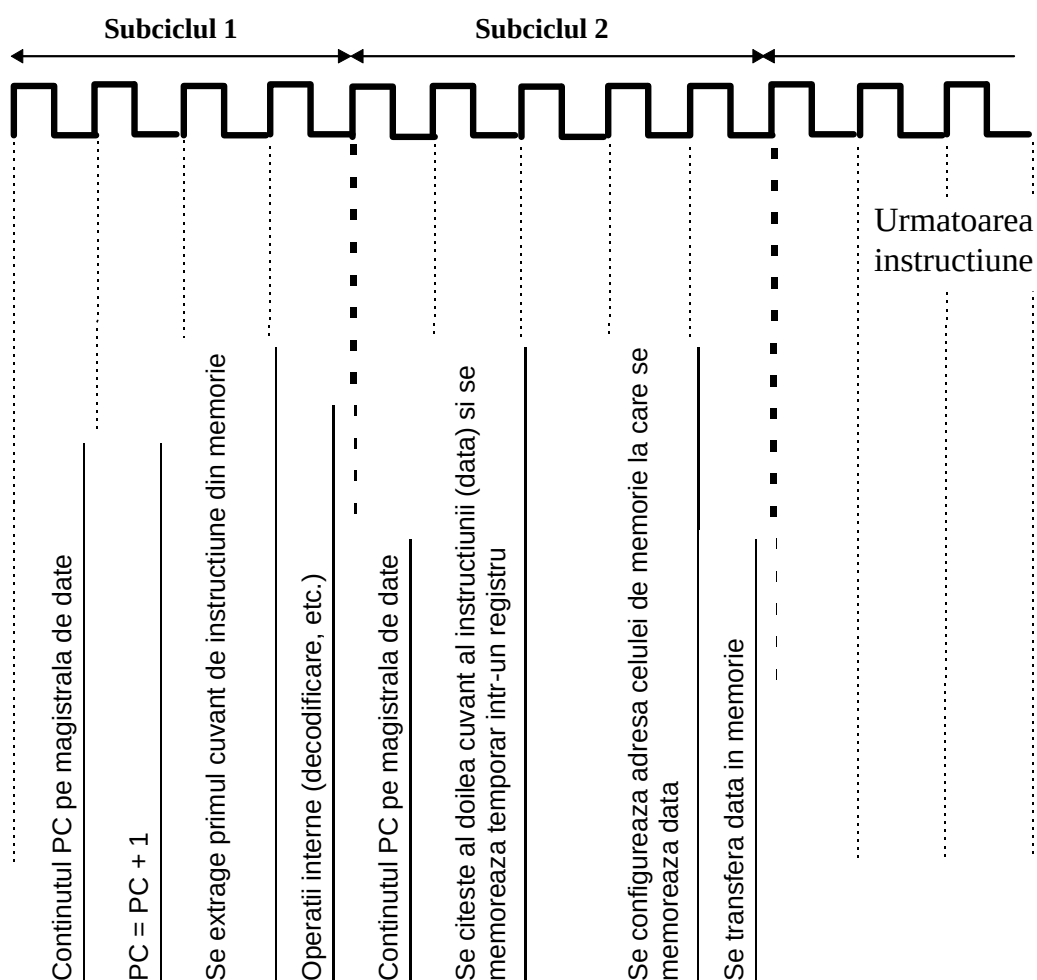


Fig.3.11. b. Ciclul microprocesorului. Exemplu: memorarea unei constante (continută în octetul al doilea al instrucțiunii)

Astfel, in figura 3.11.b se prezinta structura unei instructiuni de transfer, intr-o celula de memorie externa, a unei valori specificate in codul instructiunii respective. In primul subciclu al instructiunii se citeste si decodifica codul instructiunii; in urma acestei decodificari, sectiunea de control "stie" (din tabela de coduri microprogramate ale microprocesorului) ca trebuie citit si un al doilea cuvânt al instructiunii, continuând valoarea numerica ce se doreste a fi transferata in memoria externa. Aceasta valoare odata citita din memoria program, la inceputul subciclului al doilea al instructiunii (de la adresa de memorie indicata de catre continutul registrului *PC* - reactualizat dupa fiecare citire din memoria program), se memoreaza temporar intr-unul din registrele interne ale microprocesorului. Se trece apoi la executarea propriu-zisa a instructiunii. Astfel, se configureaza pe magistrala de adrese adresa celulei de memorie la care se va efectua transferul, iar pe magistrala de date se transmite data respectiva, din registrul temporar al microprocesorului, catre celula de memorie externa selectata. Prin secventializarea corecta a semnalelor de catre sectiunea de control (adrese, date, semnale de control), aceasta data prezenta pe magistrala de date, este inscrisa in celula de memorie externa selectata, incheindu-se astfel executia propriu-zisa a instructiunii. In continuare, o noua instructiune poate fi citita, decodificata si executata, etc. A se remarca faptul ca pe magistralele sistemului, la diferite momente de timp, se vehiculeaza informatii cu semnificatii diferite din punct de vedere functional. Astfel, la citirea cuvântului de instructiune, magistrala de adrese contine intotdeauna valoarea memorata de catre registrul contor de program, *PC*, iar datele citite de la celula de memorie astfel selectata sunt preluate de catre microprocesor si interpretate ca reprezentând cuvinte de instructiune. La executia instructiunii, in cazul instructiunilor de transfer de date externe (la / de la memorie sau interfete de intrare/iesire), pe magistrala de adrese se va utiliza adresa la/ de la care se va face transferul, iar magistrala de date va contine data transferata. Functionarea intregului sistem, dupa cum se remarca odata in plus, este coordonata de catre sectiunea de control.

Trebuie remarcat ca registrul *PC* al unui microprocesor are o capacitate (numar de biti), corelata cu adresabilitatea maxima a procesorului respectiv. Astfel, cu un registru de 16 biti, spatiul de adresare total va fi de $2^{16} = 65536$ celule de memorie. Mecanisme specifice de adresare apar la unele tipuri de microprocesoare, la care se utilizeaza combinatia mai multor registre, pentru a obtine adresa fizica dorita (de exemplu microprocesorul 8086 - pentru detalii, v.cap.4). Uzual, microprocesoarele de 8 biti au o adresabilitate de 64 ko (kiloocteti) de memorie. Microprocesoarele de 16 si 32 de biti au adresabilitatea mult sporita, de la 1 M cuvânt de memorie, la 16, 32M sau chiar mai mult.

3.3.1.2 Decodificare si control

Diagrama functionala a unei sectiuni de *CONTROL* tipice este prezentata in fig. 3.12. Revenim cu observatia ca, desi din punct de vedere fizic, registrele sectiunii de *CONTROL* apartin sectiunii de memorie a microprocesorului, functional ele fac parte din cea dintai. Aceste registre interne opereaza fara a fi vizibile (continutul lor nu poate fi examinat) sau controlabile (continutul lor nu poate fi modificat direct prin program), de catre utilizator.

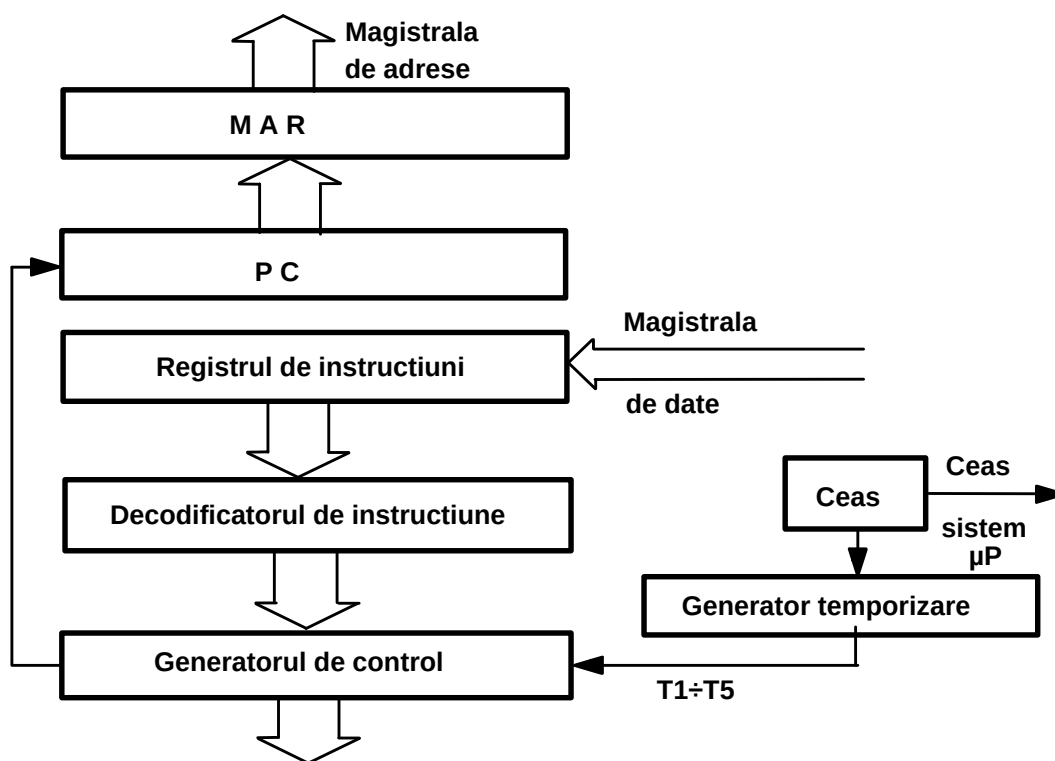


Fig 3.12. Diagrama functionala a sectiunii de control a microprocesorului

La executia *subciclurilor de fetch* (extragere a cuvântului de instructiune din memorie), dupa cum s-a specificat, continutul *PC* (reprezentand adresa celei de memorie de unde se va citi acest cuvânt de instructiune), este transferat pe magistrala de adrese a sistemului.

[Uzual, exista un registru intermediar, registru de adresa a memoriei (MAR*), care contine adresa locatiei de memorie curenta cu care se lucreaza (v.fig. 3.6, si 3.8). Aceasta adresa poate proveni fie din *PC* - la operatiile de fetch, fie dintr-unul dintre registrele generale ale microprocesorului - o adresa utilizata de catre o instructiune pentru a efectua transferul de date de la / la memorie. Registrul *MAR* este deasemenea un registru intern, inaccesibil utilizatorului in mod direct prin program].

Continutul celei de memorie adresata la operatia de fetch este transferat pe magistrala de date, si de acolo este memorat in registru de instructiune. Acesta memoreaza cuvântul de instructiune pana la decodificarea instructiunii. Uzual, dimensiunea registrului de instructiune este egala cu dimensiunea cuvântului de date al microprocesorului respectiv (8, 16 sau 32 de biti). Dupa cum se va prezenta si pentru microprocesorul 8086, unele microprocesoare au implementata o structura numita "*pipeline*" (*conducta*), care permite citirea in avans a mai multor cuvinte din memoria program, ceea ce maresc viteza globala de executie a microprocesorului (in timp ce o instructiune se executa, o alta poate fi decodificata, iar o alta poate fi citita de catre microprocesor, daca instructiunea executata nu acceseaza magistralele externe ale microprocesorului, tinand cont ca aceste operatii se desfasoara in sectiuni distincte ale sistemului microprocesor si sunt practic independente una de cealalta).

În continuare, cuvântul de instrucțiune memorat de către registrul de instrucțiune, este decodificat de către decodificatorul de instrucțiune. Acesta, în funcție de codul citit (combinația binară a acestuia), determină acțiunea specifică de executat. Decodificatorul de instrucțiune este activat doar în subciclurile în care se știe că informația de pe magistrala de date conține un cod de instrucțiune. Prin decodificarea codului de operație din primul cuvânt al instrucțiunii, secțiunea de control "știe" dacă instrucțiunea este formată dintr-unul sau mai multe cuvinte. Corespunzător, acestea se citesc succesiv, până când întreaga instrucțiune a fost transferată din memorie în microprocesor. Decodificatorul de instrucțiune convertește apoi codul operației în secvența corespunzătoare a semnalelor de control și temporizare și declanșează executia propriu-zisă a instrucțiunii (transfer de date, operații aritmetico-logice, etc.). Astfel se desfășoară, în esență, un ciclu complet al unei instrucțiuni.

Implementarea fizică a decodificatorului de instrucțiune este realizată fie prin integrare echivalentă cu o rețea de porți logice, fie cu o memorie de tip *ROM*, ale cărei adrese sunt selectate de către registrul de instrucțiune și al cărui conținut, la fiecare adresă, conține semnalele de control necesare executării respectivei instrucțiuni.

Semnalele de control ale sistemului sunt obținute în generatorul de control. Acesta, pe baza ieșirilor decodificatorului de instrucțiuni și ale ceasului sistemului, furnizează semnalele de control pentru întreg sistemul microprocesor. De asemenea, el supraveghează și registrul numărator de program (*PC*) care, după cum s-a menționat, conține adresa următoarei locații de memorie ce se va utiliza pentru citirea programului. Deoarece *PC* va trebui încărcat cu o adresă dată (de exemplu, la startul sistemului, cu adresa primei instrucțiuni din program), incrementat sau decrementat, modificat în funcție de instrucțiunea ce se execută, generatorul de control este cel mai indicat (cunoscând ce operație se execută), pentru a-i supraveghea funcționarea. Trebuie menționat aici faptul că toate tipurile de microprocesoare au un pin special, denumit **RESET**, care, prin activare, aduce sistemul într-o stare inițială unică, permițând lansarea în execuție a primei instrucțiuni a programului (în speta, conținutul de program *PC* primește o valoare fixă, cunoscută, reprezentând adresa celulei de memorie de unde se va citi această primă instrucțiune; programatorul va amplasa la această adresă începutul programului). Semnalul **RESET** determină și unele initializări interne ale microprocesorului, tot sub incidența funcției de *CONTROL*.

În funcționarea unității centrale, tot sub incidența secțiunii de *CONTROL* a acesteia, este analizat semnalul de **READY** (*gata*), furnizat microprocesorului de către o logică externă corespunzătoare. Prin testarea valorii semnalului de **READY**, prin funcția de *CONTROL*, microprocesorul poate prelunge o instrucțiune pe o durată nedefinită. Aceasta se face introducând un număr de perioade denumite de asteptare (*WAIT*) în subciclul curent, până ce semnalul de **READY** devine activ. Pe perioada de așteptare, semnalele de adrese și control (eventual date, la operația de scriere), sunt menținute nemodificate pe magistralele sistemului. Aceasta posibilitate este introdusă pentru a permite utilizarea corectă a elementelor externe microprocesorului (memorii, porturi), cu timpi de selecție și acces mai mari decât ai unității centrale. Astfel, la selecția unui element mai lent, doar atunci când informația a fost preluată de către acesta (la operații de scriere) sau când este disponibilă la ieșirile acestuia pe magistrala de date (la operații de citire), se va activa semnalul de **READY** al microprocesorului. Este sarcina proiectantului sistemului de a cunoaște viteza de lucru a

diverselor elemente ale sistemului microprocesor si de a implementa sintetizarea corecta a semnalului de *READY*. Trebuie mentionat ca orice operatie externa cipului microprocesor (inclusiv citirea instructiunilor) trebuie sa asigure generarea semnalului de *READY*. Daca memoriile si porturile sistemului au viteza de lucru suficient de rapida, permitand absenta perioadelor de asteptare, si daca programul este corect (pentru a nu se ajunge la selectii incorecte, adrese inexistente in configuratia sistemului microprocesor), se poate activa permanent semnalul de *READY*, evitand configurarea unei logici de generare a acestui semnal.

3.3.1.3 Generarea semnalelor de ceas. Temporizarea operatiilor

Toate operatiile dintr-un sistem numeric se desfasoara la intervale discrete de timp. Aceasta impune ca, pe baza unei frecvente fixe (impusa de un element exterior de tipul cristalului cu cuar), microprocesorul sau un circuit aferent specializat sa sintetizeze semnalul de ceas al sistemului. Acesta va determina viteza de baza de operare a sistemului. Temporizarile pentru fiecare subciclu al ciclului microprocesorului sunt obtinute, pe baza semnalului de ceas (prezent ca un tren continuu de impulsuri de frecventa fixa), in sectiunea de *CONTROL*, de catre generatorul de temporizare (vezi fig. 3.12). Utilizand o serie de circuite de temporizare si secventializare, acesta va coordona momentele de aparitie a diferitelor semnale de control, cat si de vehiculare a adreselor si datelor pe magistralele sistemului. Astfel, va fi posibila selectia corecta si la momentele de timp necesare a circuitelor exterioare unitatii centrale, asigurandu-se transferul corect al informatiilor in sistem. In par. 3.3.1.1 s-a prezentat o structura posibila a ciclului si a subciclorilor aferente ale unui microprocesor (v.fig. 3.11).

Frecventa normala de lucru a unui microprocesor este de ordinul megahertzilor (1--10--50) MHz (din ce in ce mai mare, la tipurile cele mai evolute de microprocesoare). Semnalul de ceas este utilizat atat intern, la sincronizarea operatiilor sectiunii de control al microprocesorului, cat si extern, pentru sincronizarea componentelor exterioare microprocesorului, cu functionarea acestuia. Dupa cum s-a precizat deja, un ciclu instructiune dureaza un numar dat de subcicluri, fiecare fiind format, la randul lui, dintr-un numar specificat de perioade de ceas elementare. Cunoscand, din manualul de utilizare al unui microprocesor, numarul si structura subciclorilor fiecarei instructiuni a acestuia, se poate calcula, pentru o frecventa a semnalului de ceas data, durata instructiunii respective. (Acele calcule pot fi necesare in cazul unor aplicatii de timp real, necesitand performante ridicate in ceea ce priveste viteza de calcul a sistemului, si deci minimizarea si optimizarea cat mai riguroasa a programului).

Deoarece viteza de lucru a diverselor circuite de intrare, iesire, memorie, *ALU*, etc., este diferita, este necesara coordonarea functionarii lor, asigurarea unor succesiuni si durate corespunzatoare ale semnalelor vehiculate pe magistralele sistemului, pentru o functionare corecta a acestuia. Dupa cum s-a precizat deja, este necesar ca aceste sectiuni, activate la un moment-dat, sa fie "*gata*" (*READY*) de a efectua corect operatiile dorite. Generatoarele de control si temporizare (v.fig. 3.12) implementeaza aceste functii, prin utilizarea unui set de circuite de secventializare si temporizare. Toate acestea vor functiona avand la baza semnalul de ceas de sistem. Desi tipul, denumirea si succesiunea exacta a acestor semnale difera de la microprocesor la microprocesor, operatiile de baza, de citire si de scriere, din memorie sau

porturi, sunt practic aceleasi, la toate tipurile de microprocesoare. Mai mult, nu exista din punct de vedere al semnalelor externe (adrese, semnale de control), nici o distinctie intre operatiile de citire a codului de instructiune (*fetch*), si citirea datelor din memorie, la executia unei instructiuni. Aceasta diferentiere se va face de catre sectiunea de control a microprocesorului, intern.

Utilizarea semnalelor de citire/scriere si coordonarea temporală a lor cu celelalte semnale din sistem, este prezentata in mod principial in continuare. Ca o remarca, se prezinta structura si secventializarile operatiilor de citire, respectiv de scriere, din/in memorie; absolut asemanatoare sunt operatiile de citire/scriere din/in interfetele de intrare/iesire ale sistemului. In esenta, semnalele de control vor fi diferite, in sensul ca exista semnale distincte de citire din memorie, sau de citire de la o interfata de intrare; similar, vor exista semnale de scriere distincte in memorie, respectiv intr-o interfata de iesire.

(a) Ciclul de citire din memorie Operatiile dintr-un ciclu de citire sunt (v.fig. 3.13):

1. Unitatea centrala plaseaza pe magistrala de adrese continutul registrului de adrese, prin intermediul circuitelor tampon de iesire de pe aceasta magistrala. Uzual, intre semnalul de ceas al sistemului si aparitia semnalelor de adrese pe magistrala va aparea o intarziere (t_{ad}).
2. Dupa o intarziere specificata, t_{acc} , se genereaza semnalul de citire a datelor;
3. Dupa un timp de acces t_{ds} , semnalele de pe magistrala de date a sistemului sunt preluate de catre microprocesor si memorate intr-un registru intern al acestuia. Se presupune ca circuitul de memorie selectat poate furniza datele la iesirea sa , dupa acest timp t_{ds} . In cazul unor memorii mai lente, se vor insera perioade de WAIT (asteptare), (v.par. 3.3.1.2), pentru ca microprocesorul sa poata citi datele corecte de la memoria selectata.

Pentru un microprocesor cu semnalul de ceas de 20MHz, valori tipice pentru t_{ad} , t_{acc} si t_{ds} sunt de 20ns, 50ns, si respectiv 20ns. Asadar, numarul de perioade de ceas (N_r) necesare pentru a implementa ciclul de citire va trebui sa satisfaca conditia (t_p = perioada semnalului de ceas al sistemului):

$$N_r \cdot t_p > t_{ad} + t_{acc} + t_{ds} = 90 \text{ ns (tipic)}.$$

Astfel, pentru $t_p = 50 \text{ ns}$, se obtine o valoare a N_r de 1,8 perioade de ceas; tinand in seama si de intarzierile introduse de catre circuitele tampon de pe magistrala de adrese, t_{ab} , si de cea introdusa de catre circuitele tampon de pe magistrala de date, t_{db} , o relatie mai corecta ar fi:

$$N_r \cdot t_p > t_{ad} + t_{acc} + t_{ds} + t_{ab} + t_{db}, \text{ ceea ce implica uzual o durata a ciclului de citire de peste 2 perioade de ceas.}$$

(b) Ciclul de scriere in memorie Operatiile efectuate intr-un ciclu de scriere in memorie sunt descrise in mod similar ciclului de citire, in figura 3.14.

In aceasta figura, s-au notat cu:

t_{as} = timpul de stabilire al adreselor (adresele trebuie sa fie stabile inainte de inceperea transferului datelor);

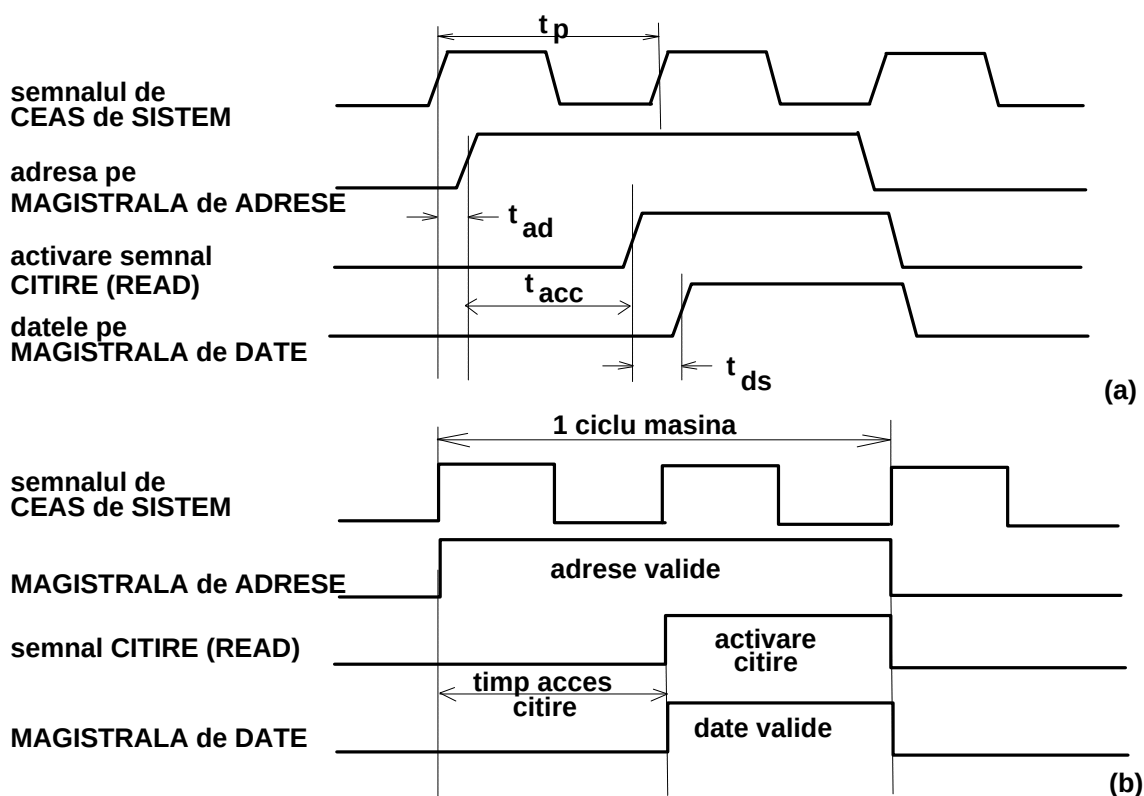


Fig.3.13. Diagramele de timp asociate unei operatii de citire din memorie. (a) forme de unda cu considerarea intarzierilor semnalelor; (b) forme de unda idealizate

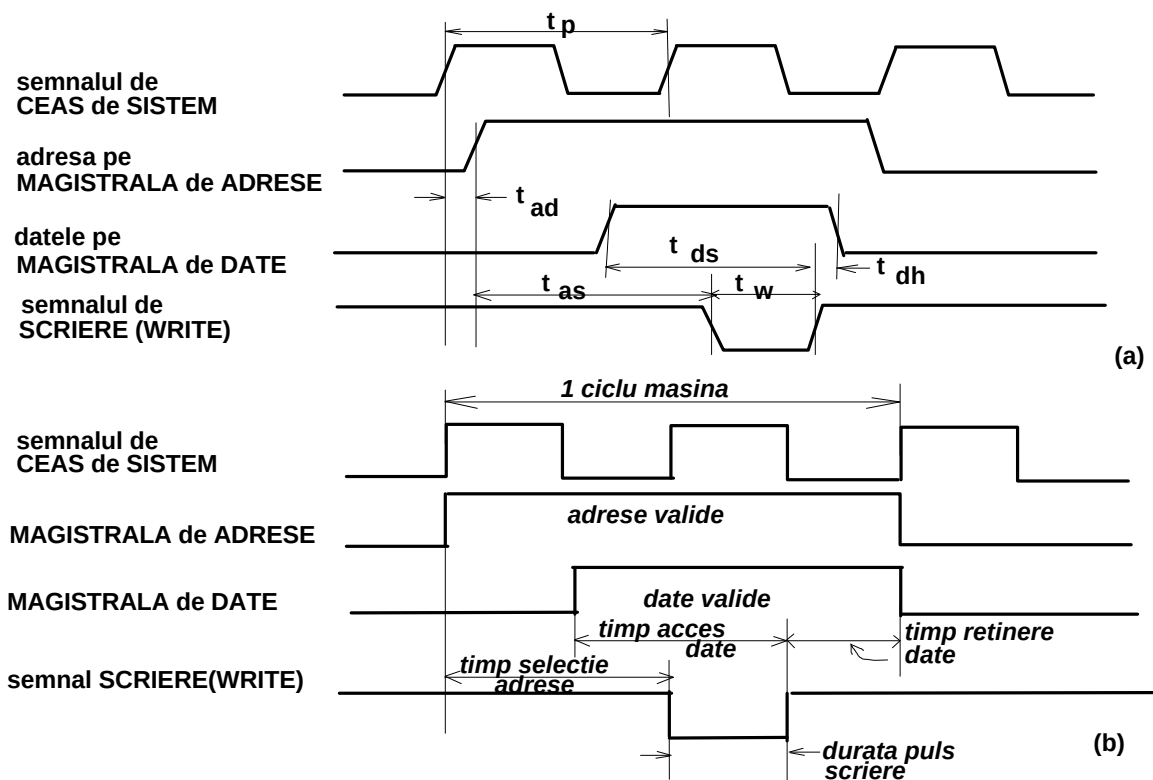


Fig.3.14. Diagramele de timp asociate unei operatii de scriere in memorie. (a) forme de unda cu considerarea intarzierilor semnalelor; (b) forme de unda idealizate

t_{ds} = timpul de stabilire al datelor, înainte de începerea transferului (datele trebuie să fie stabile deja, înainte de începerea transferului);

t_w = durata pulsului de scriere (timpul necesar pentru a înregistra efectiv datele în memorie);

t_{dh} = durata de menținere a datelor, după dispariția semnalului de scriere.

Deoarece perioada de suprapunere a semnalelor stabile, de adrese, date, și scriere trebuie să fie minimum t_w , operația de scriere va necesita un număr minim de N_w de perioade de ceas, dat de relația

$$N_w t_p > t_{as} + t_w + t_{dh} + t_{ad}.$$

În cazul unui calcul mai riguros, se obține, luând în considerare și întârzierile introduse de către circuitele tampon de pe magistralele de adrese și de date, t_{ab} , respectiv t_{db} ,

$$N_w t_p > t_{as} + t_w + t_{dh} + t_{ab} + t_{db} + t_{ad}.$$

Astfel, cunoscând valoarea diferitelor întârzieri apărute în sistem, se poate calcula durata unui ciclu complet de scriere în memorie, și numărul minim de perioade de ceas necesare.

Pentru majoritatea microprocesoarelor, ciclurile de citire sau scriere se efectuează în 2 sau 3 perioade de ceas. Numărul total de perioade de ceas necesare pentru efectuarea unei operații complete de citire sau scriere din/în memorie (sau port de intrare-iesire), este uneori denumit și **ciclu masina** (v.fig. 3.13.b și 3.14.b). Un *ciclu instructiune*, după cum s-a precizat deja, va reprezenta timpul total necesar pentru extragerea, decodificarea și executia unei instrucțiuni. Deoarece o instrucțiune poate fi reprezentată pe mai mult de un cuvânt, este necesară uneori efectuarea de mai multe cicluri de acces la memorie, la executia unui ciclu instructiune, deci a mai multor cicluri masina.

Cu cât frecvența de lucru a microprocesorului crește, vor fi necesare și componente externe din ce în ce mai rapide (memorii, porturi, cu timpi de selecție și acces cât mai reduși), pentru compatibilitate cu microprocesorul respectiv, în vederea obținerii unei performanțe maxime a sistemului (evitându-se introducerea de timpi de așteptare, *WAIT*, la efectuarea acestor accese externe).

3.3.2 Funcția de memorare

După cum s-a menționat, microprocesorul poate conține, în cazul cel mai general, atât memorie *ROM* și *RAM*, cât și, ca o categorie aparte (fizic tot memorie *RAM*), registrele.

Acestea din urmă alcatuiesc un element cu totul aparte față de memorie, din punctul de vedere al programării microprocesorului. Astfel, datele memorate de către registre pot fi manipulate, modificate, operate în *ALU*, prin instrucțiuni specifice de operare cu registrele, sau coordonat de către secțiunea de *CONTROL*. Totodată, aceste instrucțiuni sunt cele mai rapide. O operație

tipica cu un registru dureaza cateva zeci de nanosecunde, fata de sute de nanosecunde necesare atunci cand se opereaza cu memoria externa.

3.3.2.1 Registrele microprocesorului

Se pot considera doua categorii de registre ale unui microprocesor: cele **speciale** si cele **de uz general**. Cu exceptia *registriului indicator de adresa a stivei programului (stack pointer)*, registrele speciale sunt utilizate functional, in celelalte sectiuni ale microprocesorului. (De exemplu registru de instructiuni, registru adresei de program --- PC etc.).

Avand uzual o lungime de unul sau doua cuvinte de date, registrele sunt folosite la stocarea diferitelor informatii utilizate prin program, sau sub coordonarea functiei de control. Registrele de uz general sunt adresabile de catre utilizator, si pot fi folosite pentru stocarea si manevrarea datelor programului - operanzi, date, adrese de memorie continand date utile programului, etc. Uzual, aceste registre contin informatiile cele mai utilizate de catre program. Aceasta conduce la o sporire a vitezei de lucru globale a sistemului, deoarece timpii de acces la registrele interne sunt sensibil mai redusi decat timpii de acces la memoria externa a sistemului, dupa cum s-a mentionat. Aceste registre sunt conectate atat la alte registre interne ale microprocesorului, cat si la magistralele externe, prin intermediul magistrelor interne si al circuitelor tampon (v.fig. 3.15).

Registrele de uz general sunt utilizate pentru manipularea si memorarea temporara a datelor. In general, orice registru de acest tip este accesibil programatorului, prin instructiuni specifice, fiind posibile: incarcarea cu un cuvint dorit de date, deplasarea informatiei intre registre, sau intre registre si memorie, rotirea continutului acestora; tratarea unei perechi de registre ca reprezentand date pe cuvinte cu lungime dubla (la registre de 8 biti, perechi de 16 biti) si posibilitatea operarii cu acestea.

Prezentarea organizarii si a posibilitatilor specifice de utilizare a registrelor de uz general, din punctul de vedere al software-ului, va fi facuta ulterior, pe larg.

Selectarea registriului de uz general desemnat de o instructiune este facuta tot prin sectiunea de *CONTROL*. Daca, in urma decodificarii codului operatiei, se indica o operatie cu registre, se folosesc anumiti biti ai instructiunii, prin intermediul registriului de selectie (vezi fig. 3.15), pentru selectarea registriului dorit.

Numarul registrelor de uz general variaza, in functie de microprocesor, in game foarte largi, de la 2 pana la 48.

Registrele speciale - numite si registre de lucru, sau registre interne ale microprocesorului, au functii predefinite specifice in functionarea acestuia. Desi uneori pot fi setate la valori initiale specificate, aceste registre nu sunt accesibile prin program, la executia instructiunilor. Deja mentionate in aceasta categorie sunt registrele *contor de program (PC)*, *registru de instructiuni*, *registru decodificator de instructiuni*, etc.

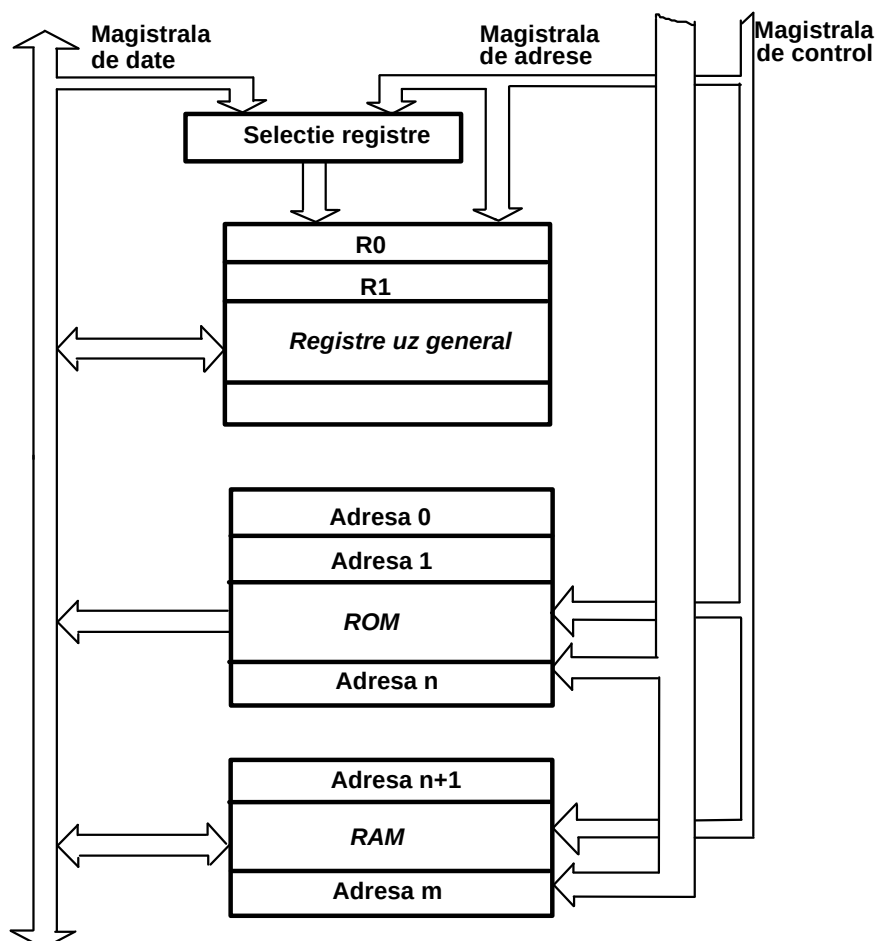


Fig.3.15. Structura principală a memoriei interne a unui microprocesor

De multe ori se utilizează subrutine la scrierea programelor, pentru o structurare și mai bună organizare a acestora. Subrutinele reprezintă porțiuni de program utilizate frecvent pentru a efectua anumite operații specifice. Subrutinele sunt apelate din diferite locuri ale programului, evitându-se astfel repetarea unei aceleiași secvențe de instrucțiuni. Deoarece la rândul ei, o subrutină poate apela o altă subrutină, etc., este necesară utilizarea unui mecanism specific pentru a se putea cunoaște adresa de revenire, la terminarea execuției unei subrutine, la punctul din program de unde se făcuse apelarea ei.

De asemenea, în funcționarea microprocesorului, pot apărea evenimente exterioare care să necesite o tratare imediată (de exemplu, necesitatea de a se citi valoarea unei marimi la intervale date de timp). În astfel de cazuri, operația curentă este oprită, iar microprocesorul trece la execuția unei secvențe speciale din program, care va efectua operațiile dorite, conform evenimentului apărut. Acest mod de a trata *evenimentele asincrone, relativ la desfășurarea curentă a programului microprocesorului, poartă numele de lucru în intreruperi*. Și la apariția unei intreruperi, în esență, se apelează o subrutină, specifică intreruperii respective.

Se va face ulterior prezentarea detaliată a funcționării microprocesorului la apelarea unei subrutine, sau la apariția unei intreruperi. Ceea ce trebuie menționat aici este faptul că o subrutină odată apelată, (din program, sau ca urmare a unei intreruperi), odată tratată, trebuie

sa permita revenirea in program, permitand continuarea acestuia din punctul de apel al subrutinei, sau de intrerupere. In acest scop exista o zona speciala a memoriei, numita **stiva (stack)**, in care se stocheaza, pe perioada executiei unei subrutine, sau tratarii unei intreruperi, adresa la care se va reveni, la terminarea acestei sectiuni speciale de program. In acest scop, microprocesorul este prevazut cu un registru special, numit **registrul indicator de adresa al stivei programului (SP - stack pointer)**. El contine adresa ultimei celule de memorie RAM, in care este salvat, la apelarea subrutinei, continutul actual al registrului contor de program (PC). Trecerea la zona de program pentru executia subrutinei se face prin modificarea valorii PC, care va contine adresa de inceput a acestei zone de program.

Microprocesorul va executa urmatoarea instructiune incepand de la aceasta adresa, deci va incepe executia subrutinei (apelata prin program sau ca urmare a unei intreruperi). La sfarsitul acestei zone de program, o instructiune specifica de revenire la programul initial va determina readucerea in PC, din memorie, de la adresa specificata de registrul indicator al adresei stivei, a valorii salvate a PC in momentul apelarii subrutinei. Astfel, programul va continua apoi din punctul de unde a fost efectuat apelul subrutinei.

3.3.2.2 Memoria RAM si ROM

Dupa cum s-a mentionat, unele microprocesoare au incluse pe cip celule de memorie RAM si ROM. Si aici, dimensiunea acestor zone de memorie este dependenta de tipul microprocesorului; de la procesoare care nu au memorie locala, la cele cu cativa ko de memorie ROM si cateva sute de octeti de memorie RAM. Adresele acestor celule de memorie sunt unice. Prin intermediul magistralei de adrese se poate selecta in mod unic o celula de memorie, sectiunea de CONTROL determinand ce anume este de facut la aceasta locatie. Daca se doreste citirea (READ) informatiei, continutul celulei de memorie apare pe magistrala de date. La o scriere a informatiei (WRITE), se memoreaza informatia de pe magistrala de date in celula selectata.

Prezentarea modului de adresare si a organizarii memoriilor exterioare va fi facuta ulterior.

3.3.3 Functia aritmetico-logica

Sectiunea care efectueaza operatiile aritmetice si logice este compusa dintr-o **unitate aritmetico-logica (ALU)**, ce efectueaza operatia propriu-zisa, si **registre speciale** necesare la manipularea si memorarea temporara a operanzilor (vezi fig. 3.16).

3.3.3.1 Unitatea aritmetico-logica (ALU)

O structura comuna a ALU este prezentata in figura 3.17. Cele doua *intrari de date* vor furniza valorile celor doi (uzual) operanzi ai unei operatii aritmetice sau logice efectuate in ALU.

Intrarea de *selectie a functiei* provine de la sectiunea de CONTROL, si specifica operatia ce se va efectua. In plus, intrarea *semnalului de transport (carry)*, este utilizata la efectuarea operatiilor aritmetice cu precizie extinsa.

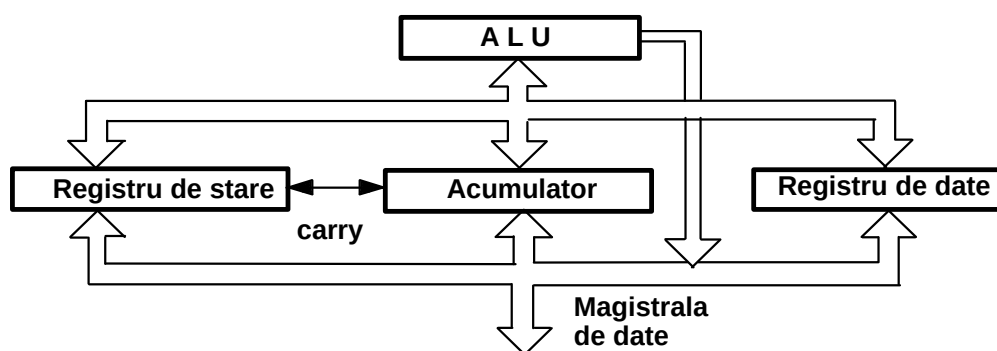


Fig.3.16. Structura secțiunii aritmetico-logice a microprocesorului

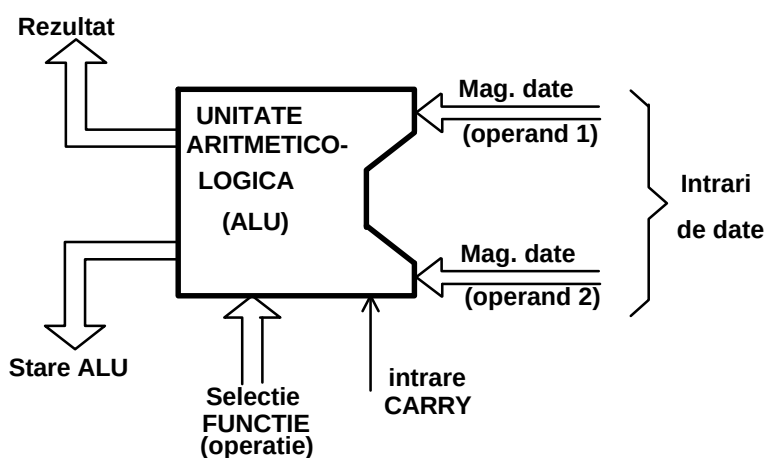


Fig.3.17. Structura de baza a ALU

Posibilitatile *ALU* depind de tipul și performanțele microprocesorului. Ele pot varia de la simple operații de adunare și scădere pe cuvinte de 8 biți și operații logice elementare, până la înmulțiri și împărțiri pe 16 biți etc. În general însă, *ALU* poate efectua adunări binare, scăderi binare, operații logice booleene, rotații la dreapta sau la stanga ale conținutului operanzilor. Alte operații, mai complexe, pot fi implementate prin succesiuni de astfel de operații simple, printr-un software adecvat. Multe microprocesoare (mai ales de 8 biți) efectuează operațiile de înmulțire și de împărțire utilizând o serie de operații elementare de adunare sau scădere. Microprocesoarele recente aparute, practic toate microprocesoarele de 16 sau 32 biți, au implementate în setul de instrucțiuni și operațiile de înmulțire și împărțire (cu numere întregi pozitive sau în complement față de doi). Evident, prețul acestor dispozitive va fi mai mare decât al procesoarelor standard. Există o singură unitate aritmetico-logică într-un microprocesor, și câte unul sau două acumuloare și registre auxiliare.

3.3.3.2 Registre pentru funcția aritmetico-logica

Una dintre cele două intrări în *ALU* este întotdeauna conectată la un registru de uz general, cu utilizare specifică, denumit **acumulator** (*A*), care deci va furniza unul dintre operanzii oricărei operații aritmetice sau logice. În plus, după efectuarea operației, în acumulator se va transfera rezultatul operației efectuate în *ALU*. Astfel, operandul aflat inițial în acumulator se va pierde. Unele operații specifice se pot efectua direct cu conținutul acumulatorului, cum ar fi

incrementarea sau decrementarea continutului acestuia cu 1, stergerea sa , etc. Fiind adresabil ca oricare alt registru general, continutul acumulatorului poate fi incarcat / transferat direct sau indirect (v.par. 3.5), din/in memoria sistemului, sau alte registre de uz general, fara a necesita utilizarea *ALU*.

Pentru memorarea temporara a celuiilalt dintre operanzi (in cazul efectuarii operatiilor care necesita doi operanzi), se prevede un **registru de date**. Datele in registrul de date si acumulator sunt vehiculate sub comanda sectiunii de *CONTROL*, dinspre / catre registrele de uz general, sau memorie, prin intermediul magistralelor de date interna, respectiv externa.

Pentru a se cunoaste caracteristicile rezultatului unei operatii (depasirea rangului cuvantului de date, la o adunare, de exemplu, semnul rezultatului, sau faptul ca s-a obtinut un rezultat nul etc.), este prevazut un registru special, **registru de stare**. Acest registru special, inaccesibil direct utilizatorului, are intre 8 si 16 biti, fiecare bit avand o semnificatie aparte, si fiind controlat de cate o structura logica specifica. Uneori acest registru este denumit si registrul *indicatorilor de conditie*. Continutul bitilor individuali, denumiti **indicatori de conditie (flag)**, este modificat (setat sau resetat), in functie de rezultatul operatiilor efectuate in *ALU*. Acesti indicatori (unii dintre ei putand fi initializati prin program, cu $1=set$, $0=reset$), reprezinta baza mecanismului de luare a deciziilor de catre unitatea centrala. Continutul registrului de stare va furniza la orice moment de timp informatii importante despre starea procesorului si rezultatele ultimei operatii aritmetice sau logice efectuate in *ALU*. Desi continutul acestui registru nu este direct accesibil prin program, executia *instructiunilor conditionate de salt sau apel de subrutina* permite testarea indirecta a valorii bitilor indicatori de stare. In functie de semnificatie, bitii de stare din registrul indicator de stare vor fi modificati prin program sau in urma efectuarii operatiilor in *ALU*. In descrierea detaliata a setului de instructiuni al unui microprocesor, se indica pentru fiecare instructiune daca si care anume dintre bitii de stare sunt modificati, si cum, de catre instructiunea respectiva. Programatorii trebuie sa cunoasca aceste specificatii, la elaborarea programelor implementate.

Semnificatia principalilor biti indicatori de stare intalniti la majoritatea microprocesoarelor este prezentata in continuare.

Carry (CF) Indicatorul de transport: este utilizat pentru a indica depasirea preciziei de lucru a microprocesorului, in urma efectuarii unei operatii aritmetice (de exemplu, adunarea a doua numere a caror suma depaseste dimensiunea cuvantului de date al procesorului). Rezultatul se va memora in aceste cazuri in acumulator, cu exceptia bitului cel mai semnificativ, memorat in *indicatorul de transport (carry)*, utilizat astfel ca o extensie a acumulatorului. Acest bit poate fi astfel ulterior testat, sau utilizat in operatii ulterioare ale *ALU* (v.fig. 3.17). Avem

$CF = 1$, daca rezultatul ultimei operatii in *ALU* a generat transport;

$CF = 0$, daca rezultatul ultimei operatii in *ALU* nu a generat transport.

Carry poate fi initializat uzual (0 sau 1), prin instructiuni specifice ale microprocesorului. Valoarea sa se va modifica insa, in cursul efectuarii de operatii in *ALU*.

Zero (ZF) Indicatorul de zero: este utilizat pentru a indica faptul ca rezultatul unei operatii

aritmetice sau logice a fost 0 sau nu. Acest bit nu poate fi initializat direct printr-o instructiune specifica. Avem

$ZF = 1$, daca s-a obtinut rezultat nul la ultima operatie aritmetica sau logica;

$ZF = 0$, daca nu s-a obtinut rezultat nul la ultima operatie aritmetica sau logica.

Paritate (PF) Indicatorul de paritate: este utilizat pentru a indica tipul sumei modulo 2 a rezultatului ultimei operatii efectuate in ALU. Astfel

$PF = 0$, daca rezultatul are un numar impar de biti cu valoarea 1;

$PF = 1$, daca rezultatul are un numar par de biti cu valoarea 1.

Acest bit este uzual testat in cadrul programelor de verificare a corectitudinii informatiilor codificate, sau la transmisii de date.

Semn (SF) Indicatorul de semn: este utilizat pentru a indica semnul rezultatului unei operatii efectuate in ALU. Astfel,

$SF = 0$, daca s-a obtinut un rezultat pozitiv (sau zero);

$SF = 1$, daca s-a obtinut un rezultat negativ.

Semnul rezultatului este decis in functie de valoarea celui mai semnificativ bit al sau, considerat ca bit de semn, numerele fiind tratate ca numere reprezentate in complement fata de doi.

Exista si alti biti de stare specifici, cum ar fi *indicatorul de activare a intreruperilor (IF)*, *de aparitie a depasirilor de calcul (overflow - OF)*, etc., numarul, tipul si semnificatia lor fiind diferite de la microprocesor la microprocesor. In capitolul 4 se va prezenta descrierea indicatorilor de conditii ai microprocesorului INTEL 8086.

In esenta, indicatorii de stare vor permite luarea unei decizii in functie de valoarea lor, permitand de exemplu, efectuarea unui salt (ramificare) in program (trecerea, din punctul in care era programul in acel moment, la o alta adresa - nu cea urmatoare - prin modificarea explicita a continutului registrului de adresa al programului (PC), se numeste salt in program), la o zona unde se va trata adecvat situatia respectiva.

3.3.4 Functia de intrare/iesire.

In general, dupa cum s-a mentionat, sectiunea de intrare/iesire este prezenta in mica masura la nivelul cipului microprocesor, cu exceptia hardware-ului necesar pentru izolarea cipului de exterior. Uzual, operatiile de intrare/iesire se fac tot prin configurarea unei adrese pe magistrala de adrese, corelata cu generarea, de catre sectiunea de control, a semnalelor adecvate de intrare sau iesire, permitand selectarea unica a unui dintre elementele de intrare/iesire (generic denumite porturi ale sistemului) si transferul in sensul dorit de la/la microprocesor la/de la port, a datelor, pe magistrala de date. (Portul poate fi un periferic, un convertor A/D sau D/A etc.).

Legarea mai multor semnale la acelasi bit al unei magistrale se va face, dupa cum s-a mai mentionat, prin *circuite tampon (buffere)*, ca in figura 3.18. Ele permit transferul de informatie

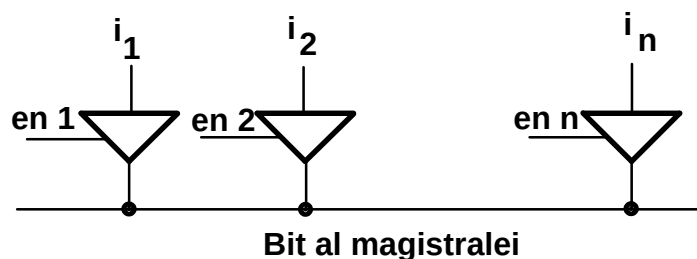


Fig.3.18. Conectarea circuitelor tampon pe magistrala

din microprocesor în afara (pe magistrala de adrese și de control), și bidirecțional, pe magistrala de date.

Aceste circuite cu 3 stări (*tri-state*), reproduc la ieșire mărimea binară de la intrare, dacă li se activează o bornă specială, de comandă (v.cap.2). Când nu sunt activate, ieșirile circuitelor *tri-state* sunt într-o stare *flotantă*

(*mare impedanță*). Aceasta permite legarea ieșirilor mai multor astfel de circuite împreună, pe fiecare linie a unei magistrale.

Selectarea unui singur *buffer* la un moment-dat va implica impunerea nivelului de zero sau unu logic de către acest circuit (în speta de către valoarea intrării lui), pe magistrala la care este conectat. Transferul de informație pe magistrale are astfel loc utilizând asemenea circuite. Pe fiecare linie (bit) a magistralelor se găsesc ieșirile circuitelor tampon uni sau bidirecționale - în funcție de tipul magistralei - ale diferitelor elemente componente ale sistemului (blocuri ale microprocesorului - pentru magistralele interne, sau microprocesor, memorii, interfețe - pentru magistralele externe).

Trebuie remarcat încă odată faptul că secțiunea de *CONTROL* răspunde de activarea corespunzătoare a circuitelor tampon ale microprocesorului. De asemenea, în perioadele când acesta nu face acces la memorie sau porturi exterioare cipului, tot secțiunea de control asigură dezactivarea circuitelor tampon (aceasta reduce consumul de putere al cipului). Selecția circuitelor tampon externe cipului microprocesor trebuie asigurată prin decodificarea corespunzătoare, unică, a semnalelor de adrese și control, pentru fiecare dintre componentele din sistem.

Dezactivarea magistralelor externe ale microprocesorului are loc și în situații speciale, când acestea sunt utilizate pentru *transferul direct de date* de la/la un periferic la/de la memoria sistemului, fără ajutorul microprocesorului (acesta fiind în așa-numită stare de *HOLD*).

3.3.5 Microprogramarea

După cum s-a precizat în secțiunile anterioare, un ciclu instrucțiune durează un număr de câteva perioade de ceas, în timpul cărora unitatea centrală va executa un set specific de operații interne, corespunzătoare instrucțiunii respective. În esență, aceste operații se clasifică într-una din următoarele subgrupe: *extragere a codului operației*, *decodificarea acestuia*, *extragere a operandului*, și *execuția instrucțiunii*. Pentru fiecare dintre aceste etape, microprocesorul execută un număr de operații elementare. Combinarea acestor operații într-o secvență coerentă determină execuția instrucțiunii respective. Aceste operații elementare, efectuate în decursul execuției unei instrucțiuni, poartă numele de microinstrucțiuni. Memorate, după cum s-a precizat deja, în zona *decodicatorului de instrucțiuni* al microprocesorului, într-o memorie

de tip *ROM*, aceste microinstrucțiuni nu sunt accesibile utilizatorului. De fapt, microprocesorul va executa o înlanțuire de asemenea microinstrucțiuni, similar unei subrutine, la executia fiecărei instrucțiuni. Înlanțuirea microinstrucțiunilor, specifică executiei fiecărei instrucțiuni, este o caracteristică a microprocesorului, și este fixă, pentru microprocesoarele uzuale, fiind implementată intern, fix, în structura hardware a acestuia. Există însă și microprocesoare speciale, (de exemplu microprocesoarele *bit-slice*), la care utilizatorul își definește singur setul de instrucțiuni, pe baza unui set de microinstrucțiuni elementare acceptate de către microprocesor. Oferind flexibilitate sporită (se poate configura un set de instrucțiuni specific, adaptat unei aplicații date), această soluție prezintă dezavantajul implementării software a setului de instrucțiuni, ceea ce reduce în esență viteza de lucru globală a sistemului. Marea majoritate a microprocesoarelor uzuale au un set de instrucțiuni fix, impus de către firma constructoare.

Figura 3.19 prezintă desfășurarea microoperațiilor într-un microprocesor fictiv, la execuția unei operații de adunare a două numere aflate în memorie, și de memorare a rezultatului într-o altă celulă de memorie. Se evidențiază operațiile necesare a fi efectuate, la executia propriu-zisă a instrucțiunii (deci după citirea și decodificarea cuvântului de instrucțiune).

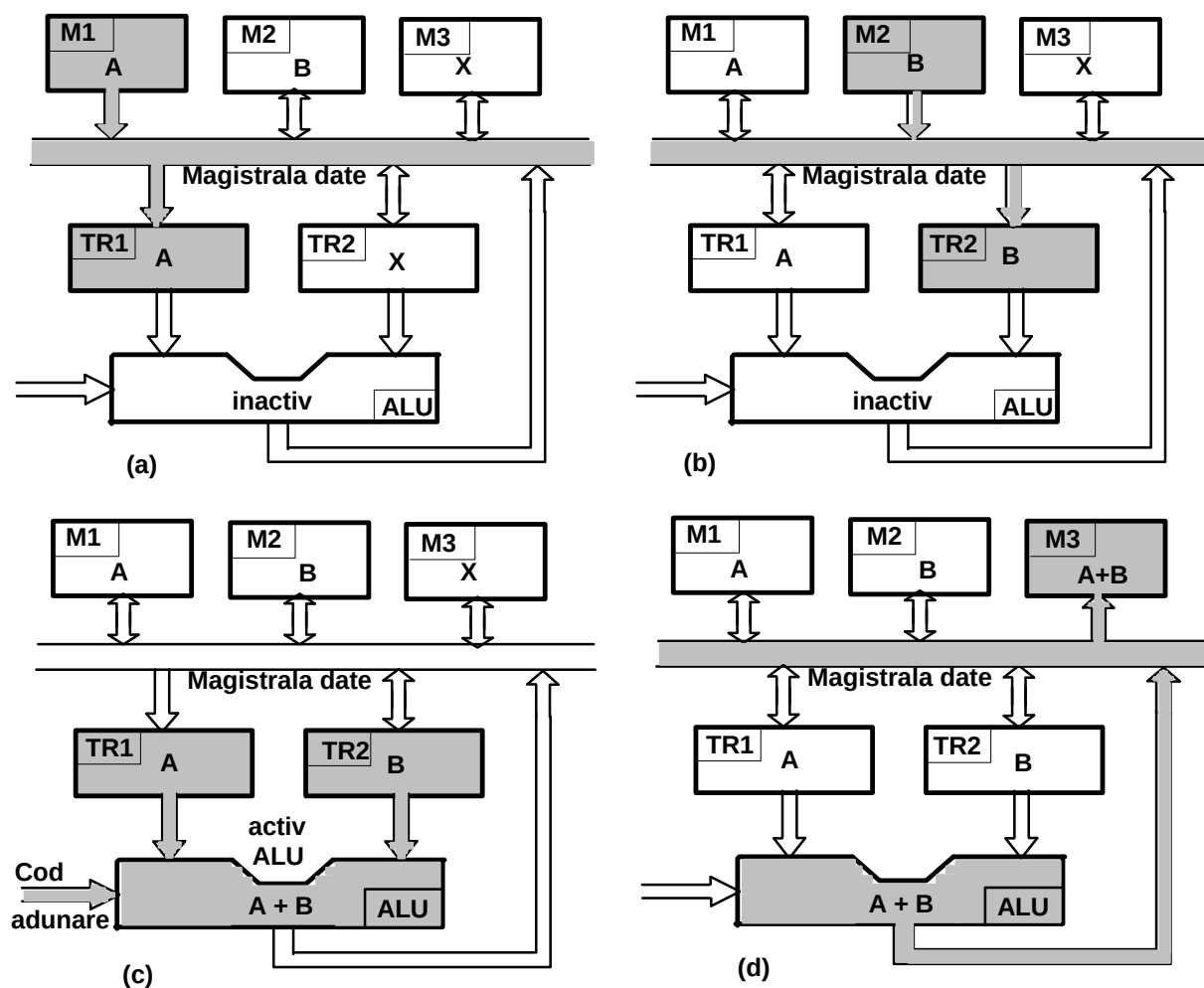


Fig. 3.19. Microinstrucțiunile executate la adunarea a două numere din memorie:
(a) citirea primului operand; (b) citirea celui de-al doilea operand;

(c) adunarea celor doua numere; (d) salvarea rezultatului in memorie

S-au notat cu TR1 si TR2 doua registre temporare, utilizate pentru stocarea operanzilor cititi din cele doua celule de memorie, M1 si M2. Rezultatul adunarii celor doua numere se memoreaza in celula de memorie de la adresa M3. Dupa cum se remarca, pentru executia acestei instructiuni, ar fi necesare 4 microinstructiuni:

- (1) citirea primului operand (A), din celula de memorie de la adresa M1 (fig. 3.19.a);
- (2) citirea celui de-al doilea operand (B), din celula de memorie de la adresa M2 (fig. 3.19.b);
- (3) efectuarea operatiei de adunare in ALU, $C = A+B$; (fig.3.19.c);
- (4) memorarea rezultatului (C) in celula de memorie de la adresa M3 (fig.3.19.d).

3.4 MEMORIA INTERNA A SISTEMULUI MICROCALCULATOR

Memoria interna a sistemului microcalculator reprezinta o parte componenta esentiala a acestuia. Ea este utilizata pentru *memorarea programelor* executate de catre microprocesor, si pentru *vehicularea datelor si rezultatelor* diferitelor operatii efectuate. Denumita *memorie principala* a sistemului, capacitatea sa depinde de tipul microprocesorului utilizat, fiind uzual de maxim 64 ko si ajungand pana la 1 Mo (sau mai mult, la noile tipuri de microprocesoare, unde sunt utilizate capacitati maxime de 8, 16, 64 M cuvinte de memorie).

*Timpul de acces*¹ al memoriei, variind intre cateva zeci, uzual sute de nanosecunde, si microsecunde, este dependent de tipul de memorie utilizat. Dupa cum s-a precizat, efectuarea operatiilor interne ale procesorului necesita timpi foarte scurti de acces la elementele de memorie locala (registre), de ordinul zecilor de ns, deci ca viteza de lucru, memoria principala a sistemului microcalculator, externa cipului microprocesor, se situeaza imediat dupa elementele de memorie interna a microprocesorului (registre, RAM, PROM de pe cip).

Ne vom referi in cele ce urmeaza la tipurile si caracteristicile acestor circuite de memorie, utilizate pentru implementarea memoriei principale, situate in afara cipului microprocesor.

Accesul la memoria principala (externa cipului microprocesor) necesita *timpi de acces* impusi de durata subciclurilor si a ciclurilor instructiune ai microprocesorului. Astfel aceasta memorie trebuie sa poata furniza/receptiona informatii la viteza cu care poate lucra microprocesorul (citire de instructiuni, date, prelucrari, memorari rezultate). Memoriile lente introduc *perioade de asteptare (WAIT)* care micsoareaza viteza globala a sistemului.

Trebuie reamintit ca unele tipuri de microprocesoare --- de exemplu INTEL 8086 --- poseda facilitatea citirii si memorarii interne a unui set de mai multe cuvinte de instructiune succesive, paralel cu desfasurarea unor operatii interne (*pipeline*). Aceasta suprapunere a

¹*Timp de acces la o memorie*: intervalul de timp intre momentul selectarii acesteia si momentul disponibilitatii la iesirea memoriei a datei cerute (la citire), respectiv momentul memorarii datei (la scriere).

operatiilor marestre viteza globala a sistemului, necesitand totodata utilizarea unor memorii mai rapide).

Acele informatii care sunt necesare relativ rar si care ocupa un volum mare de memorie sunt, in mod uzual, memorate pe elemente de memorie externe, de tipul banda sau disc magnetic, cu timpi de acces de ordinul milisecunde pana la secunde, dar cu un pret per bit de informatie memorat substantial redus.

Din punct de vedere al *accesului la memorie*, se poate defini:

- Accesul aleator, care permite operarea directa cu elementul de memorie dorit. Caracteristic accesului aleator este timpul de acces maxim identic, pentru orice celula adresata.
- Accesul secvential, care necesita o trecere ciclica prin toate locatiile anterioare celei dorite. Evident, timpul de acces este variabil, in functie de locul unde se afla celula cautata in memorie (ca exemplu tipic de memorie cu acces secvential, putem mentiona banda, caseta magnetica).

Din punct de vedere al *modului de operare*, memoriile se pot imparti in:

- Memorie RAM, continand informatii temporar memorate ce pot fi modificate oricand in cursul operarii.
- Memorie ROM, continand informatii fixe, nemodificabile, ce pot fi furnizate doar catre microprocesor (pot fi citite de catre acesta).

3.4.1 Memorii RAM

Dupa cum s-a precizat, memoria *RAM* stocheaza *informatii variabile*. Unitatea centrala, sub controlul programului, poate citi sau modifica continutul unei celule de memorie *RAM*, dupa dorinta. Memoria *RAM* poate fi de doua feluri: statica sau dinamica.

Memoria RAM dinamica Retine informatia prin incarcarea cu sarcina electrica a capacitatii portii unui tranzistor *MOS*. Deoarece sarcina acumulata pe capacitate scade in timp, prin descarcarea acesteia, informatia se va pierde daca nu se reface periodic. Aceasta operatie de *reimprospatare* (*refresh*) a memoriei *RAM dinamica*, depinde de valoarea constantei *RC* a circuitului, (uzual de 1--2 milisecunde, cu *C* de ordinul fractiunilor de picofarad) si depinde de tipul de memorie utilizat.

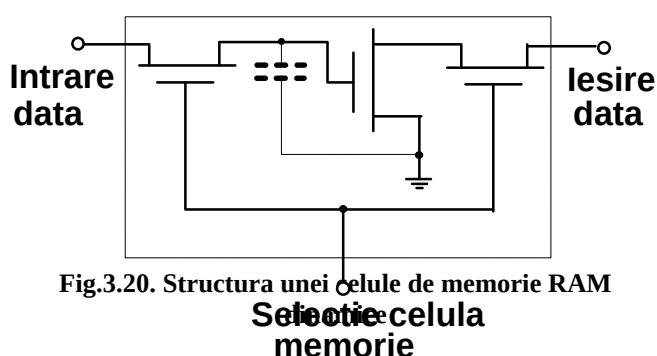


Fig.3.20. Structura unei celule de memorie RAM

Reimprospatarea are loc prin efectuarea unei operatii de citire, pe durata careia, de obicei, este oprit accesul procesorului la circuitul de memorie respectiv. Aceasta inaccesibilitate pe perioada de reimprospatare ocupa 1--5% din timpul total de lucru al memoriei dinamice.

(Structura de principiu a unei celule

tipice de memorie dinamica este prezentata in figura 3.20).

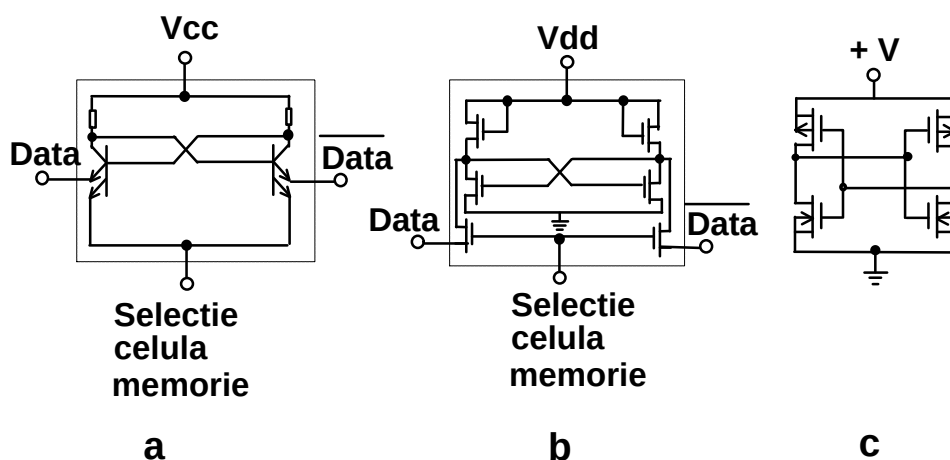


Fig.3.21. Structura unei celule de memorie RAM statice:
a) tehnologie bipolară; b) tehnologie MOS; c) tehnologie CMOS.

Memoria RAM static

Nu necesita reimprospatare; celulele de memorie sunt bistabile, similare ca schema de principiu cu bistabilele conventionale. Figura 3.21 prezinta implementarea unor celule de memorie RAM static, in diverse tehnologii.

Memoria dinamica prezinta doua mari avantaje fata de cea statica:

(a) consumul de putere mult mai redus (cand nu este adresata, consuma putere doar pe perioadele de reimprospatare); **(b)** numarul de tranzistori pe celula mai mic, deci se obtin densitati mult mai mari de celule de memorie pe cip.

Memoria statica nu necesita circuite externe speciale si nu prezinta probleme de sincronizare a ciclurilor de reimprospatare cu operatiile normale de citire/scriere ale unitatii centrale.

Deoarece memoriile *RAM* sunt memorii volatile, al caror continut se pierde odata cu pierderea tensiunii de alimentare, apare uneori necesitatea prevederii alimentarii auxiliare a acestora, de la baterii. In aceste cazuri, consumul de putere al memoriei devine un element precumpanitor in alegerea solutiei, preferandu-se memorii *RAM dinamic*, sau *RAM static* de tehnologie *CMOS*.

In functie de tip, cipurile de memorie pot fi organizate pe cuvinte de 1, 2, 4 sau 8 biti. Figura 3.22 prezinta o structura tipica a unei memorii *RAM statice*, in acest exemplu de 64×8^1 cuvinte de memorie. Pentru a selecta in mod unic un cuvant de memorie dintre cele 64, vor fi necesari ($64 = 2^6$) 6 biti de adresa.

Prin intermediul *decodicatorului intern de adresa*, acestia vor permite selectia unica a cuvantului de memorie dorit. Furnizat de catre logica de control a microprocesorului,

¹ 64×8 simbolizeaza 64 cuvinte de memorie de cate 8 biti fiecare

semnalul de $R/\overline{W}$ (citire/ $\overline{\text{scriere}}$)², va indica logicii interne a cipului de memorie sensul transferului de informatie, permitand comanda adecvata a circuitelor tampon ale memoriei, determinand totodata actiunea de inscriere a datelor prezente de pe magistrala de date, pe pinii de date ai cipului, in celula selectata, la operatia de scriere, respectiv transferarea continutului celulei selectate, pe pinii de date ai cipului, spre magistrala de date, la operatia de citire.

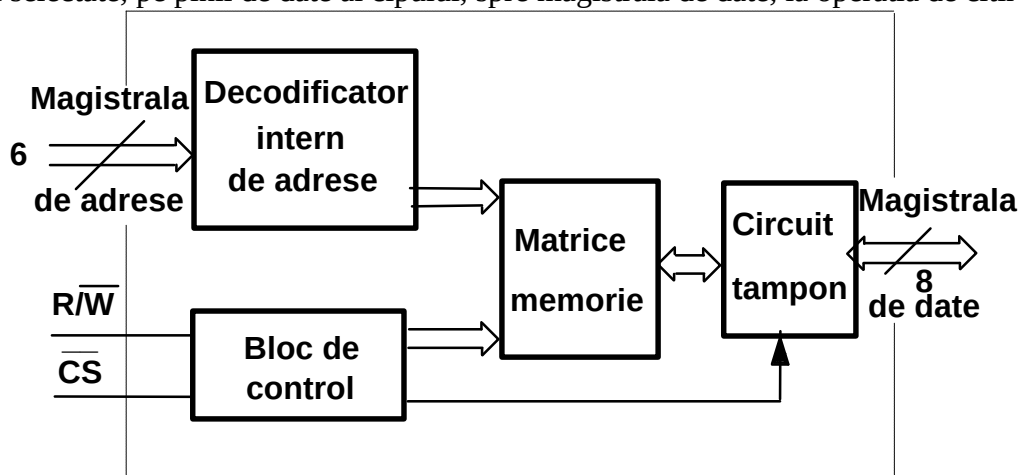


Fig.3.22. Structura unui circuit RAM static de 128 octeti

În ceea ce privește structura internă a unui asemenea circuit de memorie, o celulă elementară (bit) de memorie, de tip static (registru), este prezentată în fig. 3.23. Dacă se efectuează o operație de scriere (*WRITE*), valoarea datei de intrare (*DATA INPUT*) este memorată de către registru. Dacă se efectuează o operație de citire (*READ*), conținutul registrului este transmis la ieșirea de date (*DATA OUTPUT*), prin comanda circuitului tampon de ieșire. Deoarece operațiile de citire sau scriere nu se efectuează simultan, se utilizează uzual un singur semnal, de tipul celui din figura ($R/\overline{W}$), cu acțiuni complementare în funcție de valoarea sa logică: $1 = \text{citire}$, $0 = \text{scriere}$. Aceasta celulă de memorie stochează doar un bit de informație, deci reține doar data înscrisă la ultima operație de scriere. Totodată, operațiile de citire nu afectează valoarea înscrisă în celulă de memorie.

² se va conveni reprezentarea barată a semnalelor a căror semnificație este indeplinită când iau valoare logică "0" (active pe zero), și nebarate, când sunt active pe "1" --- de exemplu, citire/ $\overline{\text{scriere}}$ va însemna operație de citire, dacă acest semnal are valoare logică 1, și operație de scriere pentru valoare logică 0.

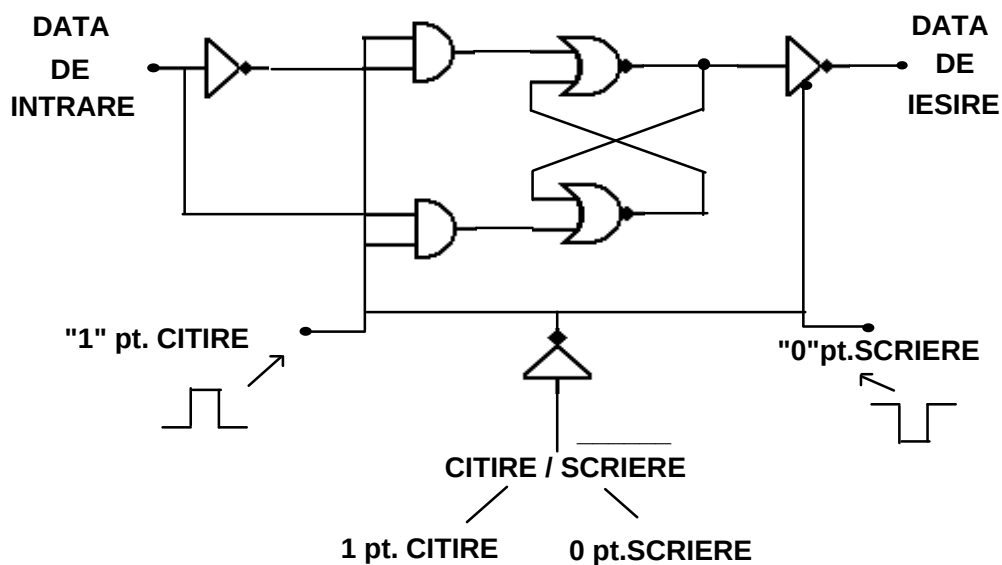


Fig. 3.23. Structura unei celule de memorie cu capacitate de un bit

Pentru a memora un cuvânt de date de n biti, (de exemplu un octet), se conectează n asemenea celule ca în figura 3.24. Aceasta permite accesul simultan la cele n celule de memorie, datele fiind înscrise sau citite în paralel din toate celulele cuvântului de date astfel configurat. Se remarcă faptul că în acest caz va fi necesar doar un singur semnal $R/\overline{W}$, aplicat tuturor celulelor de memorie simultan.

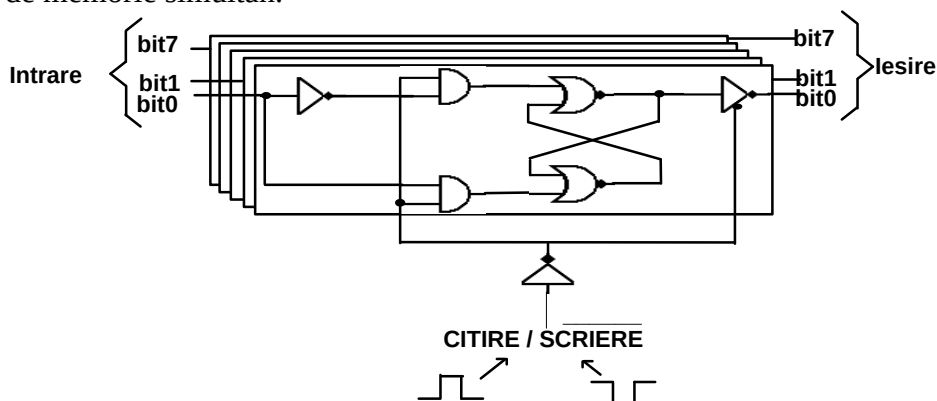


Fig. 3.24. Structura unui cuvânt de memorie RAM.

Pentru a putea configura cipuri cu capacități de memorie sporite, celulele acestora se organizează de obicei într-o matrice ca în figura 3.25, în care bitii de adresă se utilizează uzual divizat, pe *linii* și *coloane*, ce se decodifică astfel încât să se asigure, pentru fiecare adresă, selecția *unică* a unei celule de memorie a cipului.

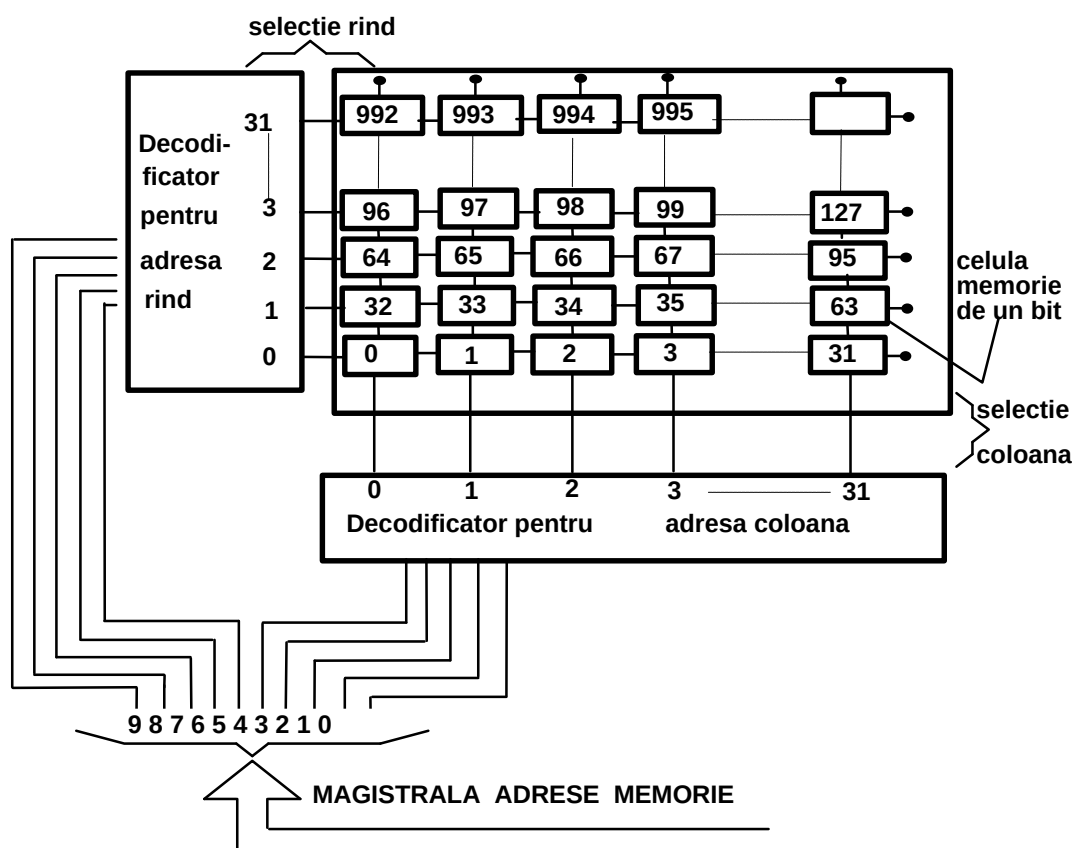


Fig. 3.25. Configuratia unui circuit de memorie cu o capacitate de 1024 celule (biti).

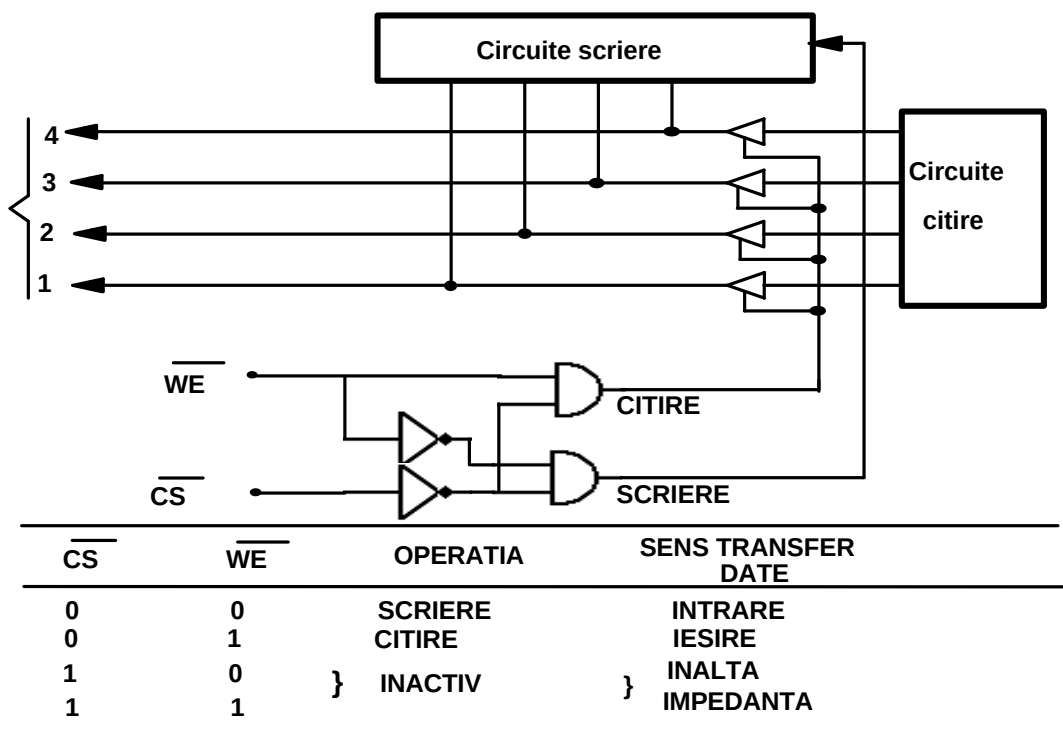


Fig. 3.26. Utilizarea tipica a semnalelor de control (citire/scriere, selectie de cip), la o memorie RAM.

Structura si utilizarea de principiu a liniilor de control ale unui circuit de memorie sunt prezentate in figura 3.26. Dupa cum se poate remarca din schema circuitului, cat si din tabela de functionare a circuitului, semnalul $R/\overline{W}$ va selecta corespunzator valorii sale circuitele de citire (pentru 0 logic) sau scriere (pentru 1 logic), care vor asigura transferul corect al datelor din/in celula de memorie selectata conform adresei prezente pe bitii de adrese ai cipului. Se mai remarca utilizarea unui pin special, $\overline{CS}$ (*Chip Select = selectie de cip*). Doar cand acest semnal este *activ* (0 logic in acest caz), este validata operatia de citire sau scriere din/intr-o celula de memorie a cipului respectiv (dupa cum se poate remarca si din tabela de functionare a cipului, din figura 3.26). Daca semnalul de selectie de cip $\overline{CS}$ este *inactiv* (1 pentru exemplul abordat), nu se efectueaza nici o operatie cu nici o celula de memorie a cipului respectiv (atat operatia de citire (*READ*) cat si cea de scriere (*WRITE*) sunt inhibitate).

Structura de principiu a memoriilor *RAM dinamice* este asemanatoare cu cea a memoriilor *RAM statice* prezentate anterior, in ceea ce priveste modul de organizare si conectare a celulelor de memorie ale cipului. In esenta, structura cipului este tot de tip matricial, celulele fiind plasate pe o retea de *linii (row)* si *coloane (column)*. In plus fata de memoriile *RAM statice*, prin modul de organizare al cipurilor de *RAM dinamic*, in vederea reducerii timpului necesar *reimprospatarii* continutului celulelor (*refresh*), ori de cate ori este selectata o linie a matricii de celule, continutul tuturor celulelor de pe acea linie este reimprospatat. Astfel, reimprospatarea se face linie cu o linie, ceea ce conduce la structuri cu un numar redus de linii, si la scheme interne speciale, utilizand tehnici de multiplexare a schemei de adresare interne a cipului (v.fig. 3.27). Bitii de adresa ai cipului se *multiplexeaza*, sub controlul unui selector bidirectional, permitand selectia unei linii si apoi, succesiv, a tuturor coloanelor matricii de memorie.

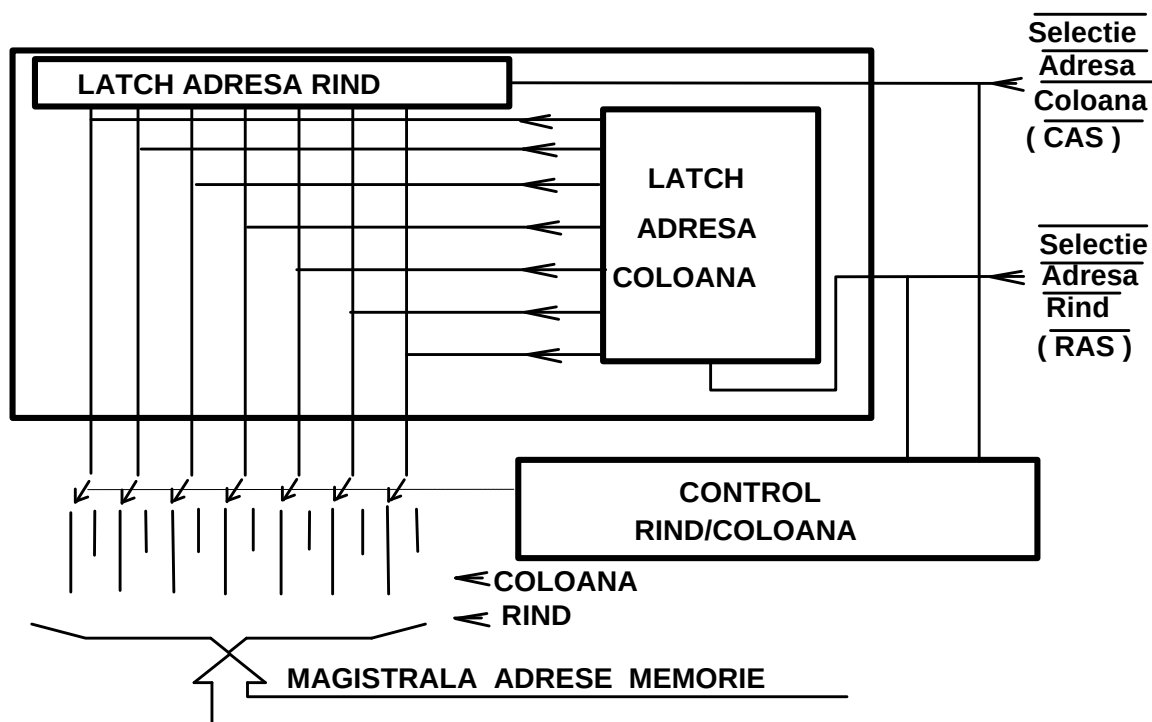


Fig.3.27. Multiplexarea liniilor si coloanelor unui circuit de memorie RAM dinamic.

Trebuie remarcat ca, in cazul memoriilor cu mai putini biti pe cuvant decat cei ai cuvantului microprocesorului, se vor conecta in paralel numarul necesar de cipuri, pentru a se obtine lungimea dorita a cuvantului de memorie.

Figura 3.28 prezinta in acest sens un exemplu de formare a unei memorii de $2k \times 8$ biti, din circuite de $2k \times 4$ biti. Se observa conectarea acelorasi semnale de adrese si control la cele doua cipuri. Semnalele de date vor fi inasa cuplate pentru primii 4 biti la primul cip, respectiv pentru urmasorii 4 biti la cel de-al doilea. O operatie cu o celula de memorie de la o anumita adresa va determina selectia simultana, in cele doua cipuri, a celulelor cu aceeasi adresa; astfel, efectuand in paralel transferul de informatie in/din ambele celule, se obtine o operatie pe 8 biti.

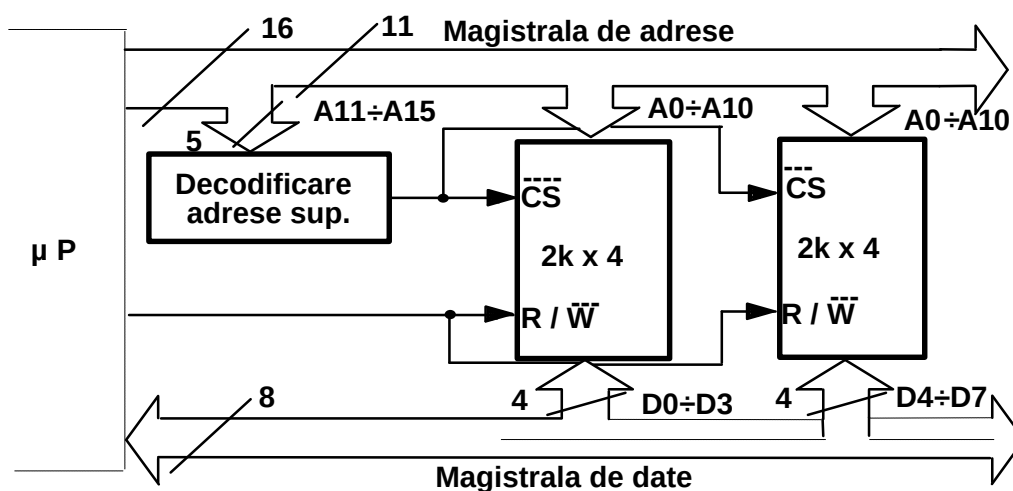


Fig.3.28. realizarea unei memorii $2k \times 8$ biti, cu doua circuite de $2k \times 4$ biti

Din punctul de vedere al microprocesorului, ansamblul celor doua cipuri de memorie vor reprezenta un singur circuit, deoarece orice operatie de citire sau scriere efectuata la una dintre adresele celulelor continute de catre cele doua cipuri va determina efectuarea transferului in *paralel*, simultan, de la/catre cele doua circuite. Conectarea distincta a acestora pe magistrala de date a sistemului va realiza operatiile pe un *cuvant de date* simultan (8 biti in cazul exemplului din fig. 3.28).

Dupa cum se poate remarca si din figura 3.28, in procesul construirii memoriei RAM sau ROM a sistemului, din mai multe elemente (cipuri), fiecare avand o anumita capacitate, uzual mai mica decat capacitatea maxima de adresare a microprocesorului, adresarea trebuie facuta la *doua nivele*. Trebuie astfel selectat intai *cipul* in care se va gasi cuvantul de memorie dorit. Prin configurarea a bitilor de adresa ai acestuia, se selecteaza apoi *celula* dorita din cip.

Primul nivel de adresare se numeste adresarea de selectie a cipului ($\overline{CS}$ - Chip Select).

Plecand de la considerentul unicitatii unei celule de memorie in cadrul sistemului microprocesor, se va proceda dupa cum urmeaza:

Se vor grupa *bitii de adresa* in doua categorii: cei *inferiori* vor fi aplicati direct bitilor de adresa ai cipurilor de memorie (astfel, pentru o memorie de $2k \times 8$, se vor utiliza 11 dintre

bitii de adresa, *direct pe cip*; $2k=2^{11}$). Restul de biti de adresa, *superiori*, vor fi utilizati intr-o logica de decodificare (vezi fig. 3.29).

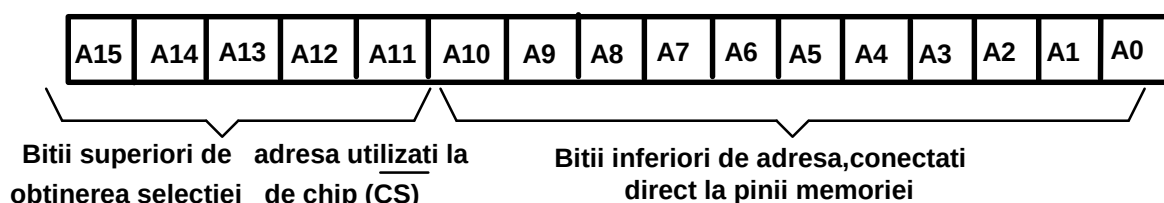


Fig.3.29. Utilizarea tipica a bitilor de adresa pentru paginarea memoriei sistemului microprocesor

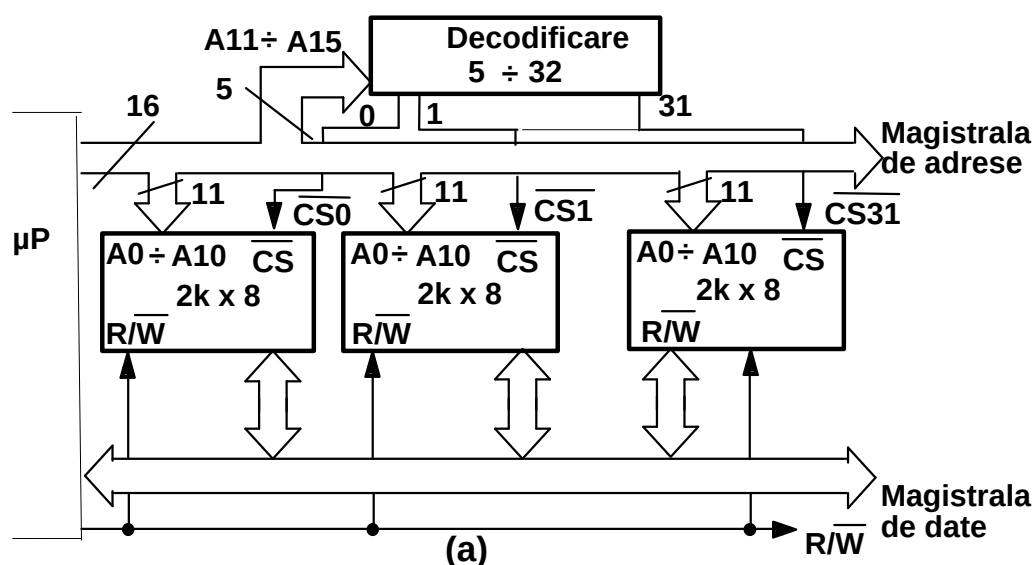
Aceasta va permite obtinerea unor semnale logice de selectie de cip ($\overline{CS}$), in numar de (fie o magistrala de 16 biti de adresa) $2^{(16-11)} = 2^5 = 32$. Se spune ca memoria s-a paginat in 32 de pagini (blocuri) de cate $2k \times 8$ cuvinte. Corespunzator unei combinatii din cele 32 posibile, data de bitii superiori de adresa, se obtine o selectie unica, la un moment-dat, a unuiu dintre cele 32 de cipuri. Acest lucru este posibil, deoarece, pe langa semnalele uzuale (biti de adresa, de date, de scriere-citire), cipurile de memorie poseda un pin special (unele tipuri de memorie, mai multi pini) de selectie de cip ($\overline{CS}$ in fig. 3.22, 3.28, 3.30.a). Doar daca pe acest pin se aplica un nivel logic dorit (in cazurile exemplificate, 0 logic¹), se va selecta, conform informatiei de adresa si control de pe pinii cipului de memorie, celula dorita. Evident, logica de decodificare trebuie sa elimine in mod obligatoriu posibilitatea selectiei simultane multiple (deci a aparitiei la doua cipuri, pentru o aceeaasi adresa, a semnalului de *cip select*). Aceasta ar determina la o citire din memorie, iesirea pe magistrala de date a informatiei din cele doua celule de memorie, expunand respectivele cipuri la riscul de a se distruge, pe de o parte, si furnizand o informatie falsa microprocesorului, deci alterand buna functionare a sistemului, pe de alta parte.

Figura 3.30.a prezinta structura de principiu a memoriei intr-un asemenea sistem. In figura 3.30.b este redată tabela de adrese ale cipurilor. Astfel, primul cip va contine celulele de memorie cu adresele intre 0 si 2047 (sau, in reprezentare hexazecimala uzual intrebuintata, de la 0 la 07FFH), al cincelea, de exemplu, adrese intre 2000H si 27FFH¹, etc. Dupa cum se observa in tabelul de configurare a semnalelor de selectie de cip din figura, indiferent de combinatia bitilor inferiori de adresa (A0÷A10) - simbolizati din acest motiv cu x² - fiecare combinatie a bitilor superiori de adresa (A11÷A15) va determina generarea cate unuiu dintre cele 32 de semnale de selectie de cip.

¹ semnalul fiind barat, implica selectia de cip pentru $\overline{CS} = 0$

¹ 1000H = 4k de memorie; 400H = 1k de memorie

² x = indiferent (don't care)



$\overline{CS}$	Configuratie adresa									
	A15	A14	A13	A12	A11	A10	A9	...	A0	
$\overline{CS0}$	0	0	0	0	0	x	x	...	x	
$\overline{CS1}$	0	0	0	0	1	x	x	...	x	
$\overline{CS2}$	0	0	0	1	0	x	x	...	x	
⋮				⋮						
$\overline{CS31}$	1	1	1	1	1	x	x	...	x	

x-adresa variabila
 0-0 logic
 1-1 logic

(b)

Fig.3.30. (a) Organizarea unei memorii de 64k x 8, pagini de cate 2k x 8 biti; (b) Configurarea adreselor. Se remarca astfel din figura modul de grupare a adreselor microprocesorului: bitii superiori (A11÷A15) sunt decodificati pentru a furniza cele 32 de selectii de cip (doar unul dintre cele 32 de semnale de selectie de cip este activ la un moment dat); bitii inferiori (A0÷A10) sunt utilizati pe pinii de adrese ai tuturor cipurilor de memorie. La efectuarea unei operatii de transfer de date de la/la o adresa de memorie, selectia de cip (data de bitii de adresa superiori), combinata cu selectia celulei interne (data de bitii de adresa inferiori), si cu semnalul de control al directiei transferului $R/\overline{W}$, vor determina selectia unica a unei celule de memorie, si deci vor asigura functionarea corecta a schemei.

Bineinteles, nu toata memoria sistemului trebuie sa fie organizata din cipuri de aceeasi capacitate. Un exemplu de organizare cu memorii diferite va fi dat in paragraful urmator.

3.4.2 Memorii ROM

Desemnând inițial memorii cu conținut fix, nealterabil, termenul de memorie *ROM* și-a îmbogățit semnificația odată cu apariția mai multor tipuri diferite de astfel de circuite.

Informația prezentă în memoriile *ROM* nu se modifică, nici în cazul unor operații de scriere accidentală în ele (operație nepermisă, de altfel), nici la dispariția tensiunii de alimentare a acestora.

Memoriile *ROM* se împart în trei grupe: prima cuprinde memoriile *ROM* programate de către producător, după cerințele utilizatorului, prin realizarea unor maste de programare specifice fiecărui set de circuite realizat. Ele reprezintă soluția finală, implementată în produse de cel puțin câteva sute de bucăți (pentru a fi economică). Orice modificare ulterioară dorită presupune comanda unui nou set de memorii, realizate adecvat.

O a doua clasă de memorii *ROM* este reprezentată de circuitele **PROM**¹, care sunt programate individual, de către utilizator. Programarea acestora este realizată prin distrugerea selectivă, conform informației ce se memorează, a unor micro-fuzibile, prin aplicarea unor tensiuni mai mari, de programare, pe bitii doriti (vezi fig. 3.31). După programare, la citirea din memorie, după mecanisme similare de adresare și selecție a celulelor, cu cele de la memoriile *RAM*, conținutul celulei selectate va fi transmis pe magistrala de date a cipului. Informația va avea valoare logică "0" pentru celulele cu fuzibilul distrus, respectiv "1" pentru celulele cu fuzibilul intact (în starea inițială, circuitul conține în toate celulele, valoarea "1" - fuzibile intacte).

Nici în cazul circuitelor *PROM*, informația odată memorată nu mai poate fi modificată.

Posibilitatea reprogramării memoriilor *ROM* este oferită de către cea de-a treia grupă de astfel de circuite, memoriile **EPROM**². Acestea rețin informația ca o sarcină într-o celulă *MOSFET*, putând fi șterse, prin expunerea cipului (acesta este prevăzut cu o "fereastră" de cuarț, în zona celulelor de memorie), la o sursă de radiații ultraviolete. După ștergere, cipul poate fi reprogramat.

¹ PROM: programmable ROM --- ROM programabil

² EPROM: erasable PROM --- memorie PROM reprogramabilă

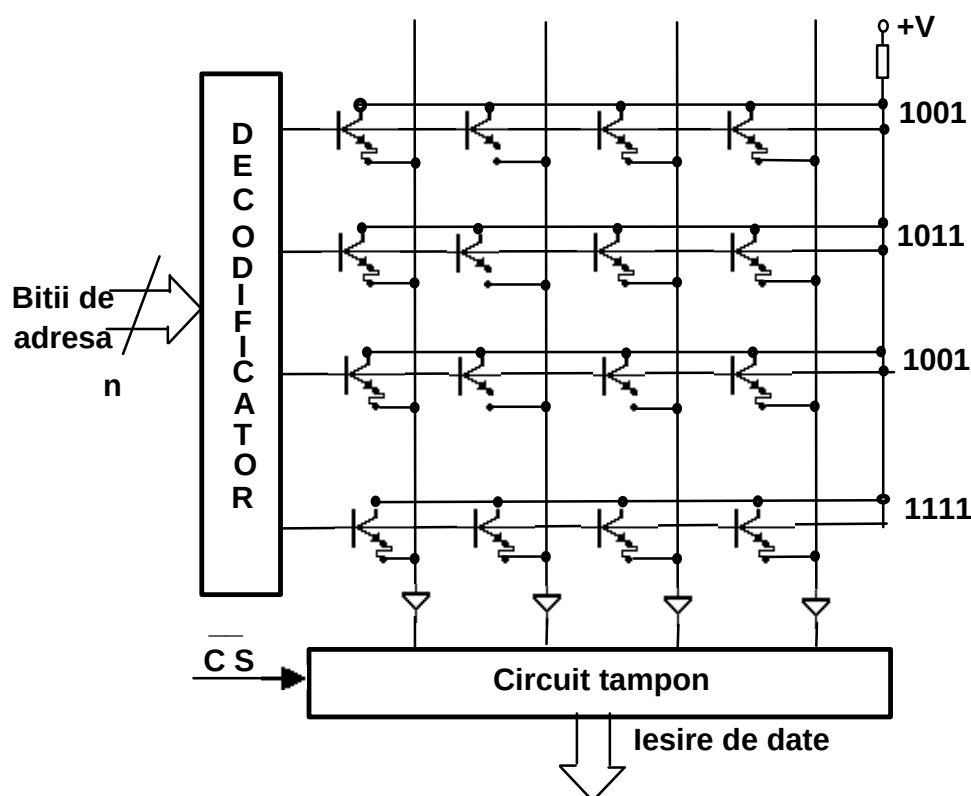


Fig.3.31. Schema de principiu a unui circuit de memorie PROM (nx4 biti).

Celulele memoriilor *EPROM* nu au conexiuni electrice (tranzistoare cu baza flotanta). Aplicarea unor pulsuri de tensiuni mai mari (25÷50 V) pe pinii de date corespunzatori, concomitent cu selectia unei celule a cipului, determina injectia unor electroni de mare energie in baza flotanta a celulei selectate, ceea ce este echivalent cu comanda tranzistorului, care astfel intra in starea de conductie. Electronii respectivi raman blocati la disparitia pulsului de programare, lasand celula respectiva programata. La aplicarea unor radiatii ultraviolete asupra cipului (prin fereastra de cuarț a acestuia), se genereaza un fotocurent intre baza flotanta a celulelor de memorie si substratul de siliciu al cipului. Aceasta readuce celulele in starea initiala, neprogramata ("1" in toate celulele).

Acest tip de memorii este utilizat cu precadere in etapa de dezvoltare si testare a sistemului sau a unei aplicatii. Timpul de pastrare a informatiilor in memoriile *EPROM* este de ordinul anilor, chiar al zecilor de ani. Operatia de stergere si reprogramare a acestor circuite poate fi repetata de zeci, chiar sute de ori. Procesul de programare a memoriilor *EPROM* se face cu ajutorul unui montaj special, denumit programator de *EPROM*-uri, montat in si coordonat de catre un microcalculator de uz general (sistem de dezvoltare, v. cap. 3.5), care va coordona procesul de programare, transmitand datele ce se inscriu in memoria *EPROM*.

O versiune mai noua de circuite reprogramabile este oferita de catre memoriile **EEPROM**¹ sau de catre memoriile **flash**. Acestea permit modificarea dorita - reprogramarea informatiei (in general selectiva), prin utilizarea unor semnale electrice, in locul radiatiei ultraviolete.

¹ EEPROM: electrically erasable PROM --- PROM ce poate fi sters electric

Astfel, modificările pot fi facute chiar în cursul execuției programului, ceea ce mărește sensibil utilitatea acestor circuite (valori ale unor parametri modificați de către utilizatorul sistemului, sau parametri de control ai unui proces, determinați în urma unor teste, sau parametrii de stare ai unui sistem, etc.), pot fi salvați în aceste memorii nevolatile; după oprirea alimentării sistemului - deconectare, sau avarii - la realimentarea acestuia, valorile salvate pot fi citite din aceste memorii și utilizate la reluarea programului, utilizând ultimele valori, actualizate, ale acestor parametri).

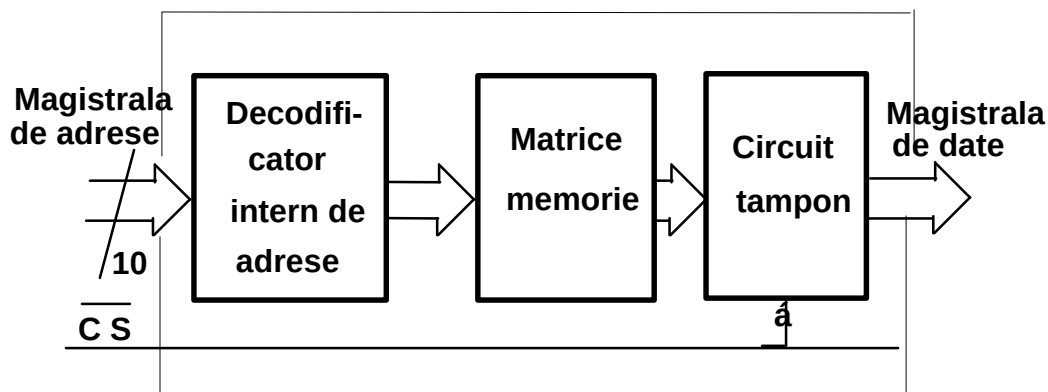


Fig.3.32. Structura unui circuit de memorare *PROM/EPROM* de 1kx8 biti.

Figura 3.32 prezintă structura principală a unei memorii *PROM* de 1kx8 biti. Se remarcă dispariția pinului de $R/\overline{W}$ (operația de scriere neavând sens). De aceea, proiectantul sistemului microprocesor trebuie să asigure formarea selecției de cip ($\overline{CS}$), atât din bitii superiori de adresă cât și din semnalul de citire din memorie. Dacă s-ar folosi doar semnalele de adresă la obținerea semnalului $\overline{CS}$, informația din celula de memorie selectată ar fi transmisă la ieșirea cipului, pe magistrala de date, și în cazul unei operații de scriere; în acest caz, microprocesorul ar transmite de asemenea informații (date) pe magistrala de date. S-ar ajunge astfel la un conflict pe magistrala de date, între semnalele provenind de la cele două surse (memorie și microprocesor). Pe de o parte, informația pe care microprocesorul ar încerca s-o înscrie într-o celulă de memorie nu s-ar înscrie niciunde (memoria *PROM* neputând fi modificată fără aplicarea tensiunilor speciale de programare), iar pe de altă parte, conflictul de date de pe magistrală poate duce la distrugerea una dintre circuitele componente (memoria sau microprocesorul).

Este **absolut interzis**, într-o proiectare corectă, să existe posibilitatea ca, printr-o operație *software* (deci prin program), indiferent care ar fi această operație, să poată apărea o situație ce poate pune în pericol integritatea *hardware* a sistemului. Utilizarea, alături de bitii superiori de adresă, a semnalului de control de citire (*READ*), pentru sinteza semnalului de selecție de cip ($\overline{CS}$), va elimina asemenea situații inacceptabile.

Uzual, memoriile *ROM* sunt organizate pe cuvinte de câte 8 biti. Capacitatea de memorare, având valori curente de 2ko, poate crește până la 32 ko, și chiar mai mult.

Gama de aplicatii a acestor circuite, practic nelimitata, cuprinde memorare de programe ale sistemului microprocesor, generare de coduri pentru caractere alfanumerice si simboluri grafice, trecerea dintr-un cod intr-altul, memorare de tabele de constante, valori tabelate ale unor functii necesare aplicatiei, etc.

Dupa cum s-a precizat in capitolul 1, utilizarea memoriilor *PROM* (cat si a circuitelor *PLD* --- ce contin si elemente de memorie *RAM* si se programeaza pentru implementarea unor functii logice diverse) permite inlocuirea tot mai accentuata a functiilor implementate prin circuite logice traditionale.

Se remarca deasemenea solutia unora dintre producatorii de circuite integrate *VLSI*, de a implementa pe acelasi cip atat elemente de memorie, cat si porturi de intrare/iesire programabile, circuite de ceas programabile etc., imbogatind posibilitatile de utilizare ale circuitului.

3.4.3 Organizarea tipica a memoriei sistemului microprocesor

Structura uzuala a memoriei unui sistem microprocesor cuprinde atat memorii *RAM* cat si memorii *ROM*. Organizarea si dimensiunile acestora depind de aplicatia in care se utilizeaza sistemul. Zonele de adrese ocupate de memoria *RAM* si *ROM* depind de proiectantul sistemului, de regula memoria *ROM* continand adresa la care trebuie sa se gaseasca prima instructiune din program, la care microprocesorul face acces dupa primirea unui semnal de *RESET*.

Pentru exemplificare, consideram o structura continand o memorie de 64 ko, organizata in 60 ko memorie *PROM* si 4 ko memorie *RAM* (fig. 3.33).

Se considera utilizarea unor cipuri de 4 ko memorie *PROM* si de 512 octeti memorie *RAM*. Se remarca utilizarea ultimilor 4 biti de adresa superiori ($A_{15} \div A_{12}$) pentru impartirea memoriei in blocuri de 4 ko.

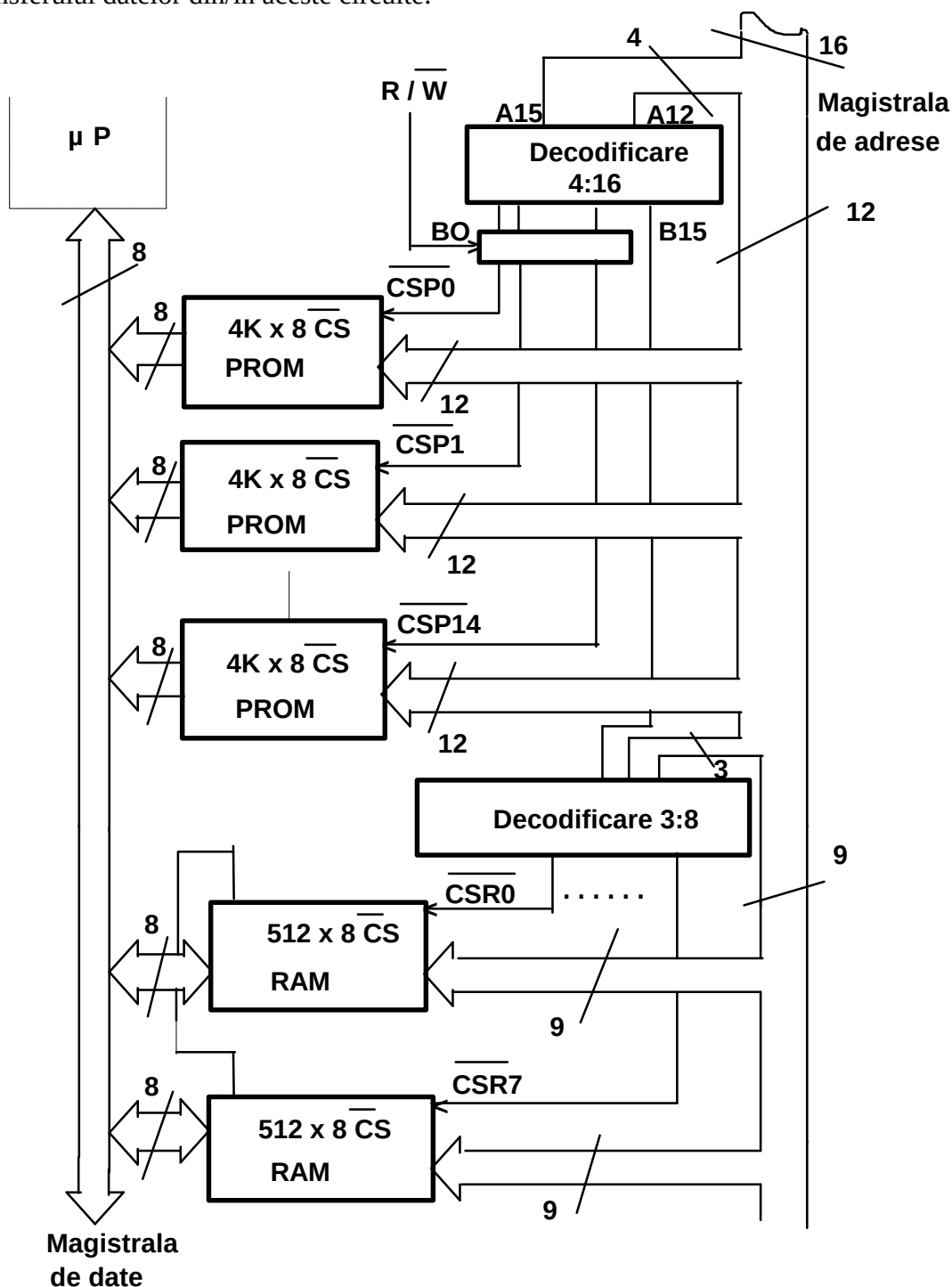
Primele 15 dintre cele 16 combinatii posibile ale acestor biti de adrese sunt utilizate pentru selectia memoriei *PROM* (generarea celor 15 semnale $\overline{CSP0}$ la $\overline{CSP14}$, de $\overline{CS}$ ale celor 15 cipuri de memorie *PROM*), bitii inferiori de adresa $A_{11} \div A_0$ fiind aplicati direct pe pinii de adresa ai memoriilor *PROM*.

Ultimul semnal de selectie, B_{15} , impreuna cu urmatorii 3 biti de adresa ($A_{11} \div A_9$), permite selectia cipurilor de memorie *RAM* [care au cate 512 octeti, deci 9 biti de adresa direct pe cip ($A_8 \div A_0$), si 7 pentru obtinerea selectiei de cip ($A_{15} \div A_9$)] - semnalele $\overline{CSR0}$ la $\overline{CSR7}$.

Asadar, memoria *PROM* ocupa primii 60 ko ai memoriei sistemului, iar memoria *RAM*, ultimii 4 ko. Se remarca utilizarea decodificarii in doua etape a adreselor superioare, pentru memoria *RAM*. Memoria este astfel organizata in 15 pagini de cate 4 ko pentru *PROM*, si 8

pagini de câte 512 octeți, situate în pagina a 16-a de 4 ko, a memoriei globale, pentru memoria RAM.

Se remarcă de asemenea utilizarea semnalului de $R/\overline{W}$ pentru validarea semnalelor de selecție a circuitelor PROM, doar pentru operații de citire (READ) din memorie, respectiv aplicarea acestui semnal direct pe cipurile de memorie RAM, în vederea selecției corecte a sensului transferului datelor din/in aceste circuite.



3.5 SOFTWARE-UL SISTEMULUI MICROPROCESOR-ASPECTE GENERALE

În acest paragraf se va aborda microprocesorul dintr-un alt punct de vedere, cel al programării sale, al posibilităților și al modului de organizare a setului de instrucțiuni. După cum s-a precizat deja, întreprinderea celor două aspecte, *hardware* și *software*, impune stăpânirea noțiunilor de bază corespunzătoare, atât în proiectarea sistemului, cât și în realizarea programelor pentru o aplicație dată.

3.5.1 Considerații de bază

Pentru o gamă foarte largă de microprocesoare, informația este dată sub forma cuvintelor de 8 biți, sau a cuvintelor de 16 biți. Deoarece o instrucțiune poate necesita un număr mai mare de biți pentru transmiterea informației necesare executării unei anumite operații, instrucțiunile unui microprocesor vor avea o lungime de unul sau mai multe cuvinte de 8 (16, 32) biți. Întotdeauna primul cuvânt din instrucțiune va conține codul operației (eventual continuat și în al doilea cuvânt), următoarele cuvinte continuând, dacă sunt necesare, eventuale valori numerice semnificând constante sau adrese de celule de memorie sau porturi, cu care operează instrucțiunea respectivă.

Citind primul cuvânt al instrucțiunii, microprocesorul îl decodifică și, sub incidența secțiunii de *CONTROL*, execută operația corespunzătoare. Dacă, de exemplu, codul operației indică o instrucțiune cu trei octeți (fie un microprocesor de 8 biți), în care octeții 2 și 3 conțin o adresă din memorie, al cărei conținut va fi adus în acumulator, succesiunea de operații ce au loc va fi după cum urmează:

- se citește primul octet al instrucțiunii;
- se decodifică codul operației;
- conform codului, se mai citesc doi octeți suplimentari (la fiecare citire, PC este incrementat), de la adresele succesive primei citiri (instrucțiunea are cuvintele unul după altul în memoria program);
- se configurează pe magistrala de adrese adresa desemnată de către cei doi octeți suplimentari citiți și se citește această celulă de memorie, conținutul ei fiind transferat în acumulator;
- se trece la citirea și executarea altei instrucțiuni.

În general, pentru o organizare pe cuvinte de 8 biți, cele 256 de combinații posibile oferite de un cuvânt sunt suficiente pentru a se configura un set de instrucțiuni destul de bogat. Evident, microprocesoarele de 16 biți au o paletă de posibilități mult crescută.

Astfel, de exemplu, figura 3.34 prezintă structura principală a instrucțiunilor unui microprocesor pe 8 biți, având cuvinte de instrucțiune de 1, 2 sau 3 octeți. După cum s-a precizat, primul cuvânt conține codul operației, și eventual, operandul (operanzii) instrucțiunii, cuvântul (cuvintele) suplimentare (dacă există, în funcție de instrucțiune), continuând date necesare la executia instrucțiunii respective.

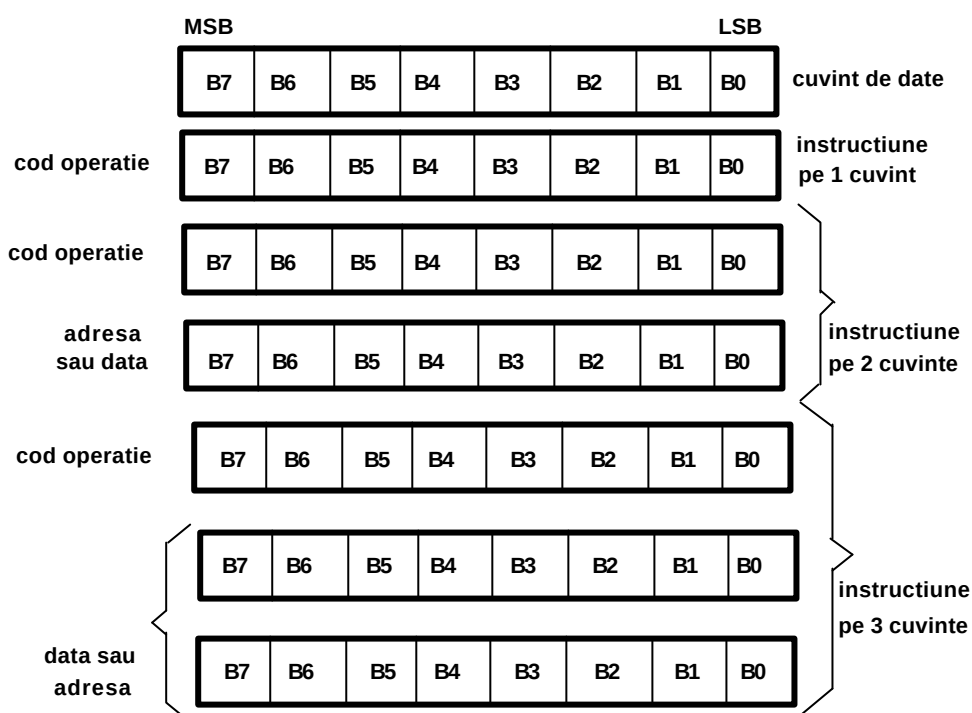


Fig.3.34. Structura principală a instrucțiunilor unui microprocesor

3.5.2 Moduri de adresare

O operație frecventă în diversele etape de execuție ale unui program este referirea la diversele celule de memorie RAM, ROM, sau la registrele microprocesorului. Aceste referiri la locații de memorie implică *operația de adresare*, fiind posibile multiple **moduri de adresare**. Indicat prin codul operației ce se execută, *modul de adresare* implică obținerea, prin diverse procedee, a adresei cu care se va opera în instrucțiunea respectivă.

Există o gamă relativ importantă de moduri de adresare. În funcție de tipul microprocesorului, se implementează prin setul de instrucțiuni unele dintre aceste moduri de adresare. O posibilitate de apreciere a performanțelor unui microprocesor este dată și de paleta de moduri de adresare ale acestuia. Cu cât gradul de sofisticare oferit de modurile de adresare crește, cresc și posibilitățile și flexibilitatea programatorului de a realiza programe mai eficiente, compacte, performante. Pe de altă parte, intervin deasemenea, ca de altfel în general în procesul elaborării programelor, și elemente subiective, legate de opțiunile personale ale programatorului, de experiența și capacitatea sa de a găsi soluția optimă pentru implementarea unei aplicații date.

Vom prezenta în continuare cele mai întâlnite moduri de adresare, utilizate la diferite tipuri de microprocesoare, însoțite de explicații și exemplificarea utilizării lor. În esență, se va explica felul în care pentru fiecare mod de adresare în parte, se obține operandul unei instrucțiuni a microprocesorului. Unele instrucțiuni contin și acest operand, altele indică adresa sau modul de obținere al adresei unde se va găsi operandul respectiv. Mecanismul obținerii acestor operanți va diferenția modurile de adresare posibile. Figura 3.35. reprezintă în mod sintetic cele mai des întâlnite

moduri de adresare. Deși figura exemplifică aceste moduri de adresare pentru un microprocesor de 8 biți, prezentarea ce urmează este general valabilă, și pentru microprocesoare cu capacități sporite (16 sau 32 de biți).

Exemplele prezentate utilizează instrucțiuni și notații specifice microprocesorului INTEL 8086. Pentru a nu intra în detalii care la acest stadiu al prezentării pot fi incomode și pot complica inutil prezentarea, se prezintă în esență modul de obținere al *deplasamentului* în cazul adresării memoriei sistemului microprocesor. Mecanismul de operare al microprocesorului 8086 pentru obținerea *adreselor efective*, prin utilizarea registrelor de segment, în combinație cu deplasamentul, nu este evidențiat aici, el fiind de altfel transparent pentru scrierea programului, atâta timp cât se vor utiliza registrele segment implicit destinate fiecărei instrucțiuni în parte.

Detalii suplimentare se pot găsi la prezentarea microprocesorului 8086, făcută în capitolul 4 al lucrării.

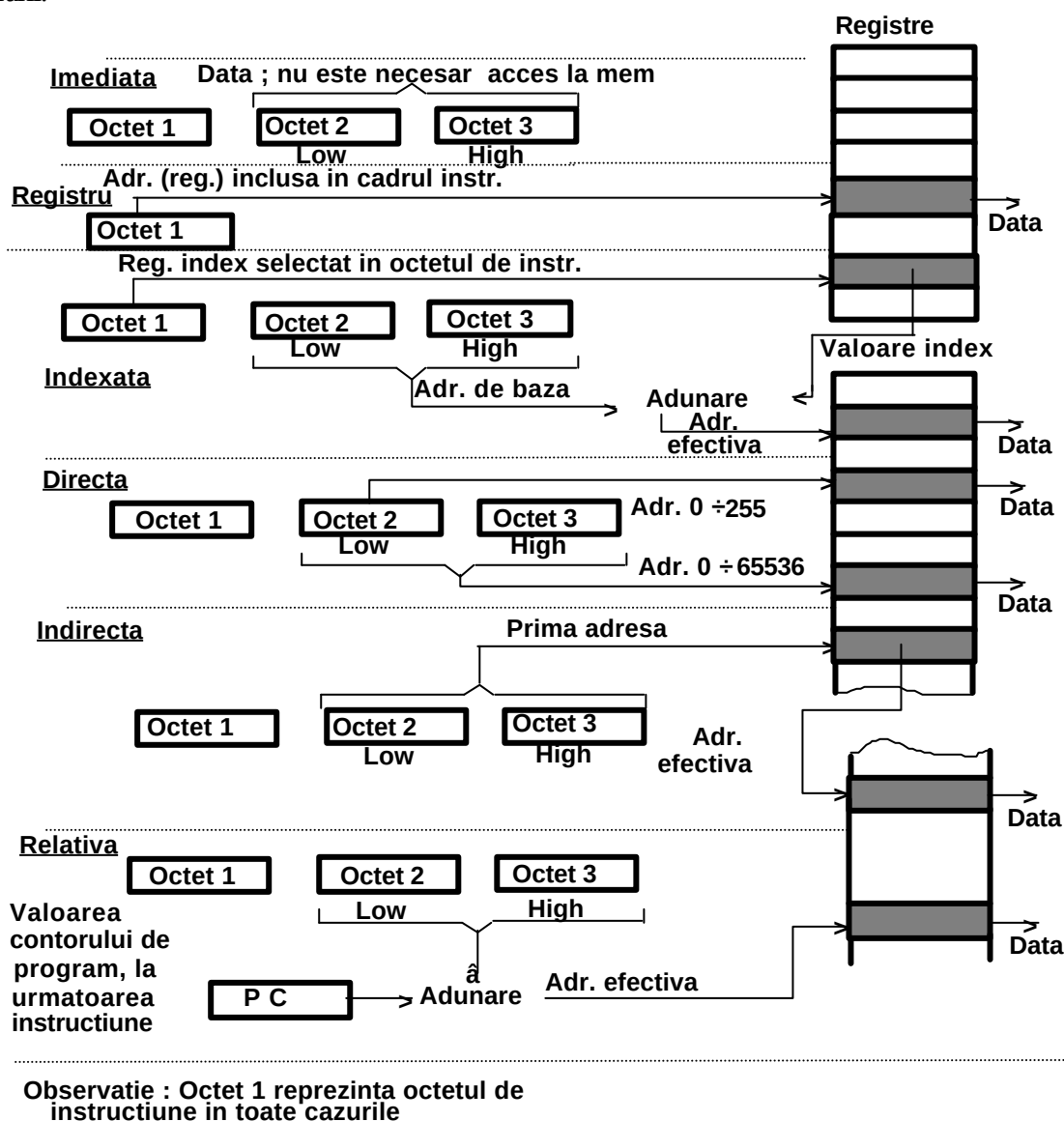
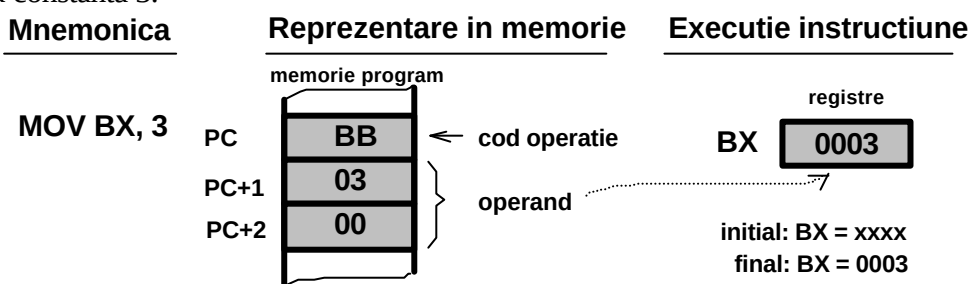


Fig.3.35. Moduri de adresare ale unui microprocesor

Adresarea imediata

În cazul *adresării imediate*, instrucțiunea conține în primul cuvânt codul operației, iar în următorul cuvânt (următoarele cuvinte), conține o valoare constantă, reprezentând chiar operandul necesar pentru executia instrucțiunii respective (v. fig. 3.35). Aceasta valoare este deci definită încă din faza de scriere a programului, fiind dată de programator în codul instrucțiunii, reprezentând deci în general valori constante, fixe, ale programului. Acest mod de adresare este cel mai simplu mod de manipulare a unei valori constante de către programator, necesitând un timp redus de execuție, deoarece imediat după citirea codului operației, în subciclul următor al instrucțiunii, se extrage (*fetch*) valoarea operandului, utilizat de către instrucțiune la executia sa.

Adresarea imediata este utilă în compararea conținutului unui registru cu o valoare constantă, în vederea luării unei decizii în program, fără a mai necesita utilizarea de memorie RAM suplimentară. Principala limitare a acestui mod de adresare este dată de faptul că valoarea operandului, fiind conținută în codul instrucțiunii, este fixă pentru un program dat, neputând fi modificată în cursul rulării acestuia. De exemplu, o instrucțiune de transfer a unei valori constante într-un registru al microprocesorului 8086 va avea structura din figura 3.36, indicând încărcarea registrului BX cu valoarea constantă 3.

Fig. 3.36. Executia unei instrucțiuni ce utilizează adresarea imediată.**Adresarea directă (absolută)**

În cazul *adresării directe*, instrucțiunea conține, după primul cuvânt, al codului operației, un cuvânt (sau 2, în funcție de tipul microprocesorului și de capacitatea sa de adresare), reprezentând *adresa efectivă* la care se găsește operandul necesar pentru implementarea instrucțiunii respective (v. fig. 3.35). Astfel, dacă s-ar utiliza adrese reprezentate pe un octet, cu un cuvânt se poate adresa oricare dintre 256 de celule (prea puțin, uzual); dacă se utilizează 2 octeți pentru reprezentarea unei adrese (reprezentate pe 2 cuvinte de instrucțiune pentru microprocesoarele de 8 biți, sau pe 1 cuvânt de instrucțiune pentru microprocesoarele de 16 biți), se poate indica în instrucțiune, direct, adresa uneia dintre 64 k cuvinte de memorie. Pentru un microprocesor cu capacitatea totală de adresare de 64 k cuvinte, sau pentru adresarea în interiorul unei pagini de memorie de 64 k cuvinte (specifica unor microprocesoare de 16 biți, precum INTEL 8086), reprezentarea pe 16 biți a adresei, în cadrul cuvântului de instrucțiune, permite programatorului adresarea oricăreia dintre celulele de memorie adresabile (global, sau în pagina de memorie curentă). Figura 3.37. prezintă un exemplu de

instrucțiune ce utilizează adresarea directă, pentru a transfera conținutul unui registru (AX = 0523H), într-o celulă de memorie de adresă specificată (ALFA = 07F8H), din segmentul de date curent (celula conține inițial o valoare în locul căreia se va înscrive noul operand, conținutul registrului AX).

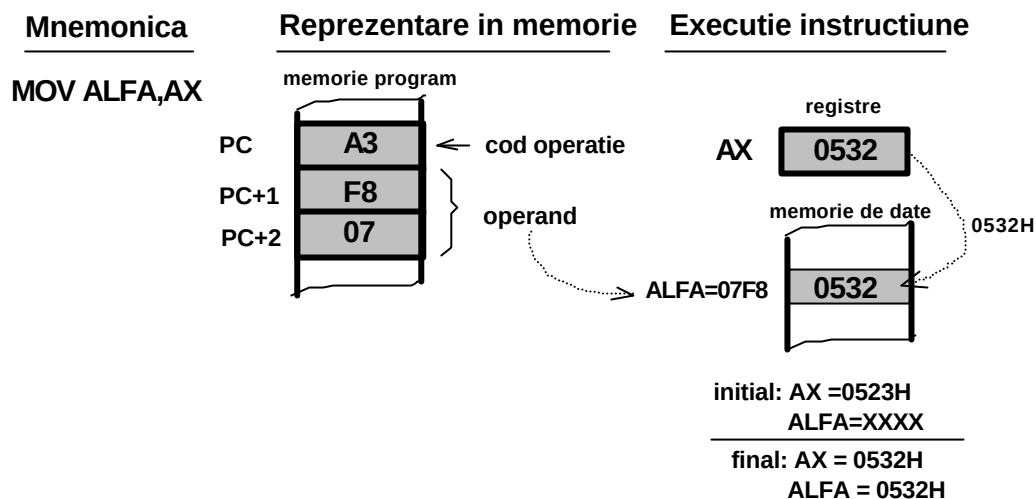


Fig. 3.37. Execuția unei instrucțiuni ce utilizează adresarea directă

Avantajul *adresării absolute* față de *adresarea imediată* rezultă din modul de obținere a datei utilizate la execuția instrucțiunii: după cum s-a specificat deja, la *adresarea imediată*, structura fixă a instrucțiunii implică utilizarea unei valori constante a operandului; la *adresarea absolută*, dacă se modifică conținutul celulei de memorie a cărei adresă este conținută în instrucțiune, instrucțiunea va utiliza la execuție date de la adresa respectivă, care poate diferi pe parcursul execuției programului. Ca un exemplu, figura 3.38 prezintă comparativ două secvențe de program, prima efectuând adunarea a două numere constante, specificate în codul operațiilor (utilizând *adresarea imediată*), iar cea de-a doua efectuând adunarea a două numere aflate în două celule din memoria de date (utilizând *adresarea directă*).

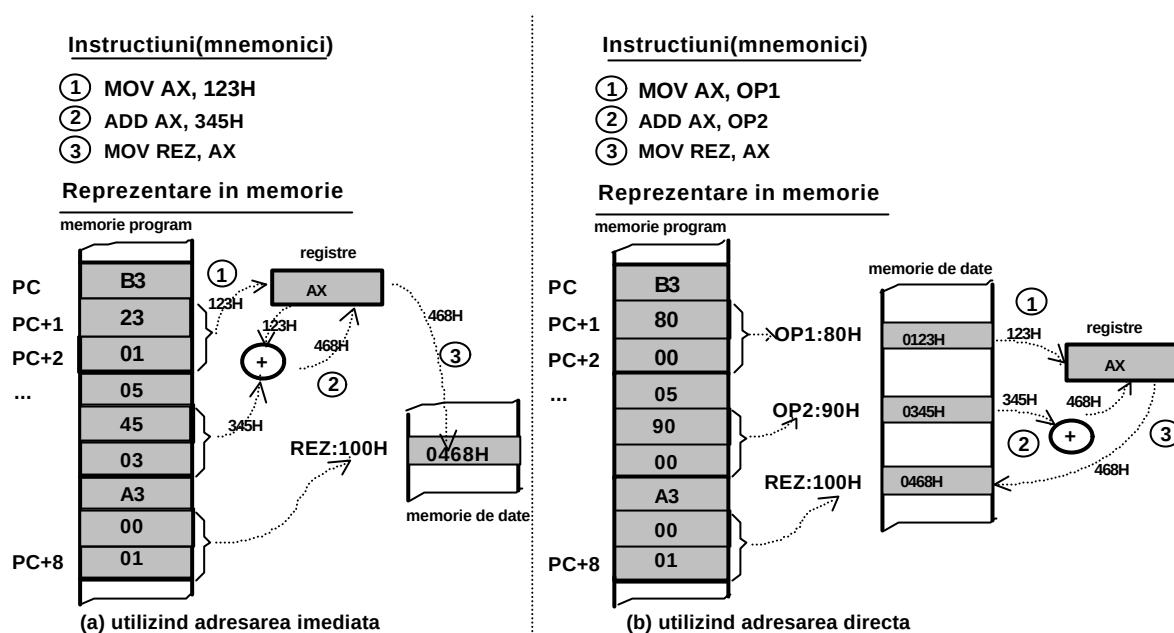


Fig. 3.38. Compararea implementarii programelor utilizand adresarea imediata (a), sau directa (b)

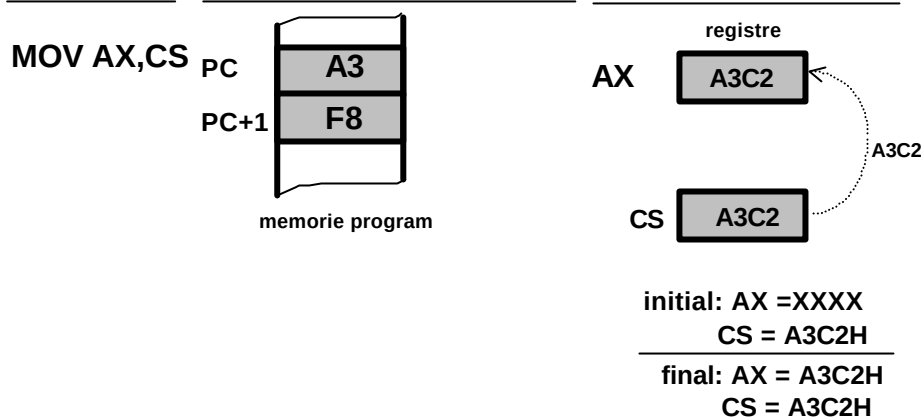
In esenta, pentru a modifica datele ce se aduna, programatorul va trebui sa modifice, in cazul *adresarii imediate*, codul instructiunilor (deci programul), pe cand in cel de-al doilea caz, al *adresarii absolute*, programatorul va trebui sa modifice continutul celulelor de memorie din *memoria de date*, in care se gasesc cele doua valori; programul ramane neschimbat in acest caz, pentru diferitele valori ale celor doua numere din memoria de date. Solutia *adresarii imediate* se va prefera in cazul operarii cu constante (ce nu se schimba in timpul programului), fiind o solutie mai rapida la executia programului.

Adresarea registru

Acest mod de adresare reprezinta o varianta a *adresarii directe*, in care operandul (operanzii) sunt continuti in *registrele* de uz general ale microprocesorului, in loc de memorie (v.fig. 3.35). Deoarece numarul de registre de uz general ale unui microprocesor este relativ redus, se codifica adresa registrului (registrelor) pe cate un cimp de 3-4 biti (functie de numarul maxim al acestor registre), continut in insasi cuvantul de instructiune, care astfel contine operanzii instructiunii. Aceasta permite ca instructiunea sa aiba doar un singur cuvânt, fiind deci scurta, si sa se execute foarte rapid (datorita vitezei de executie sporite, la accesarea registrelor interne ale microprocesorului, dupa cum s-a mai mentionat).

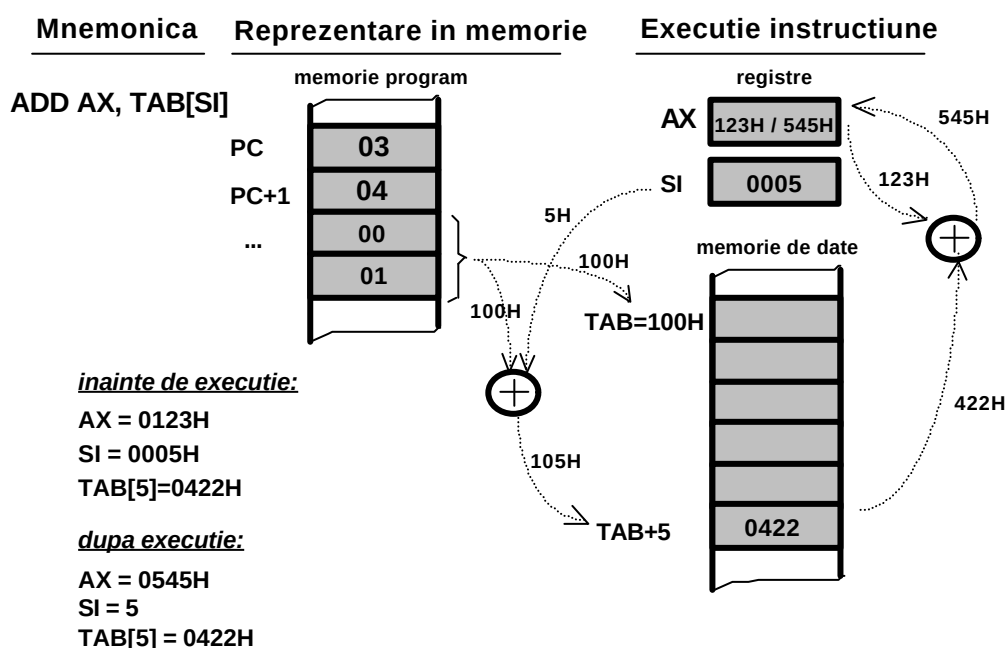
Figura 3.39 prezinta un exemplu de instructiune utilizand adresarea registru, care transfera continutul unuia dintre registrele microprocesorului (CS), in alt registru al microprocesorului (AX).

Mnemonică Reprezentare în memorie Executie instructiune

Fig.3.39. Executia unei instructiuni ce utilizeaza adresarea registru

Adresarea indexata

Unele microprocesoare sunt prevazute cu registre denumite registre index, care pot fi utilizate pentru implementarea acestui mod de adresare. In esenta, adresa datei ce se va utiliza de catre instructiune se obtine prin adunarea continutului registrului index, specificat de catre instructiune, cu o valoare numerica continuta in codul instructiunii. Aceasta permite ca, prin modificarea continutului registrului index, sa se obtina adrese diferite ale datei utilizate de catre instructiune. Figura 3.40 prezinta o instructiune de adunare la continutul acumulatorului (AX), a valorii continute in celula de memorie cu adresa data de suma dintre adresa de inceput a unui tabel (TAB) si un indice continut intr-un registru index (SI).

Fig. 3.40. Executia unei instructiuni ce utilizeaza adresarea indexata.

Continutul *registrului index* poate fi modificat automat (prin indicarea în codul instrucțiunii a acestui lucru), *incrementat* sau *decrementat*, pregătindu-l astfel pentru utilizări ulterioare, ale aceleiași instrucțiuni (de exemplu într-o buclă de program), pentru a se adresa locații distincte (uzual consecutive), ale memoriei. Tipic, acest mod de adresare este util la adresarea unor elemente ale unui tabel de valori. Adresa de început a tabelului (*adresa de baza*), este conținută în *codul instrucțiunii* (ca operand al acesteia), iar deplasamentul față de începutul tabelului (deci *indicele* elementului de tabel ce se dorește a se adresa), în *registrul index*.

Astfel, prin implementarea unei bucle de program, se pot adresa în mod succesiv locațiile unei zone de memorie. La fiecare trecere a programului prin buclă respectivă, conținutul *registrului index* se reactualizează (prin incrementare, de exemplu), fiind pregătit pentru a adresa următorul element al tabloului. Figura 5.41 prezintă un asemenea exemplu, în care buclă de program prezentată se execută de un număr de ori egal cu valoarea inițială a registrului CX. La fiecare pas, valoarea *registrului index* (SI) - inițial 0 - este incrementată, determinând adunarea succesivă a tuturor numerelor din tabelul TAB (fie 10 numere). La ieșirea din buclă, registrul AX (setat inițial cu 0), va conține suma conținuturilor tuturor celulelor de memorie din tabelul TAB.

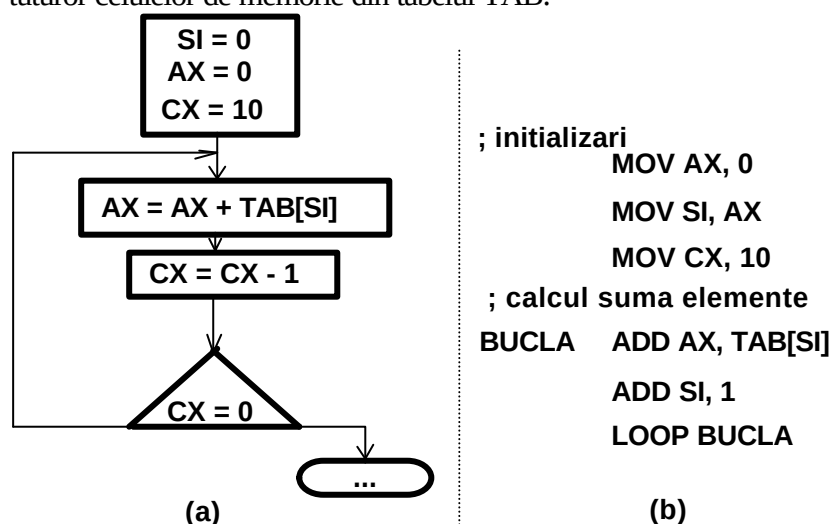


Fig. 3.41. Implementarea unei bucle de program, ce utilizează adresarea indexată
(a) Schema logică a secvenței de program; (b) secvența de program

Adresarea indirectă

În esență, în cadrul *adresării indirecte*, instrucțiunea conține în cuvântul de instrucțiune, ca operand, o adresă la care se va găsi, în memorie, adresa efectivă unde este localizată datea cu care va opera instrucțiunea la executia sa (v.fig. 3.35). Acest tip de adresare oferă un grad suplimentar de flexibilitate față de adresarea indexată, în sensul că apar "două grade de libertate" în ceea ce privește modul de localizare a datei ce se manipulează în instrucțiunea respectivă. Ca o variantă, la adresarea indirectă prin registru, instrucțiunea se referă la un registru al microprocesorului (sau la o pereche de registre), care conține adresa datei ce se va utiliza în instrucțiune. Figura 3.42 conține un exemplu

de acest tip, în care se schimbă conținutul registrului BX cu conținutul celulei de memorie a cărei adresă este conținută în registrul SI.

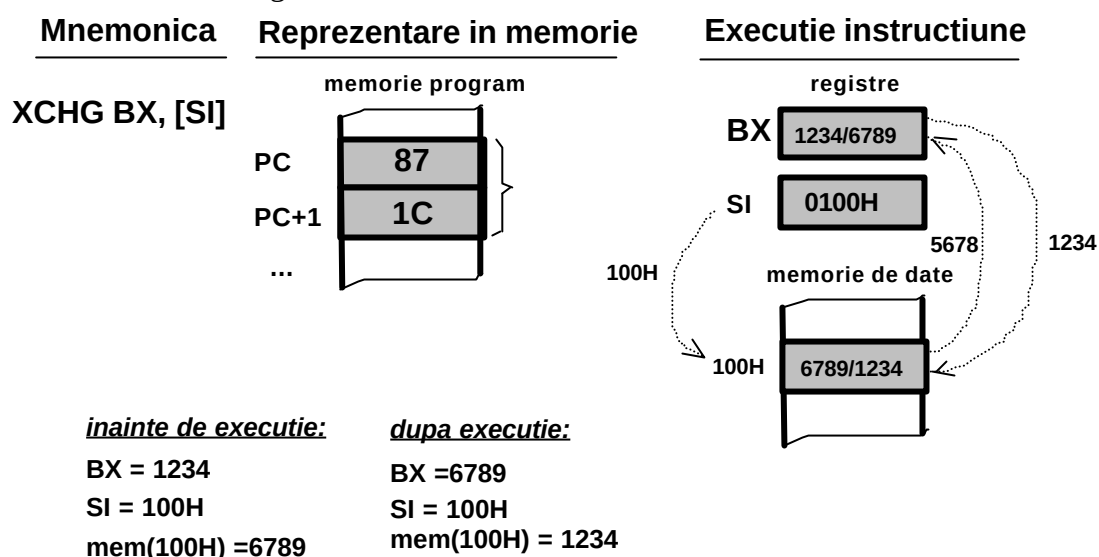


Fig.3.42. Execuția unei instrucțiuni ce utilizează adresarea indirectă

Adresarea relativă

Acest mod de adresare utilizează operandul conținut în codul de instrucțiune pentru a efectua o deplasare "în sus" sau "în jos" în program, din punctul curent (de la adresa curentă, conținută în registrul *contor de program*), cu valoarea acestui *operand*, deci cu un număr dat de cuvinte de memorie. Uzual, acest tip de adresare se utilizează la implementarea instrucțiunilor de *salt relativ în program* (*salturi scurte*). Operandul va reprezenta în acest caz nu o adresă, ci o constantă (pozitivă sau negativă - un număr în complement față de 2), reprezentând numărul de cuvinte de instrucțiune peste care trebuie să sara programul "înainte" în memorie - în sensul creșterii adresei (pentru valori pozitive ale operandului), sau "înapoi" în memorie - în sensul scăderii adresei (pentru valori negative ale operandului), de la locația curentă a programului. Această valoare conținută în operandul unei instrucțiuni cu adresare relativă se numește "offset" (*deplasare*), reprezentând *diferența* dintre adresa instrucțiunii destinație și cea a instrucțiunii imediat următoare instrucțiunii curente. Aceste valori de deplasare sunt uzual calculate de către programul asamblor, în mod automat, programatorul utilizând o scriere formală, convenabilă.

Figura 3.43 prezintă modul de scriere, și codul generat pentru o secvență de program ce utilizează o adresare relativă înainte (JLE ERR), și una înapoi (JMP TST). Se poate remarca modul de obținere al deplasamentului în cele două cazuri, reprezentat în cod complementar față de 2.

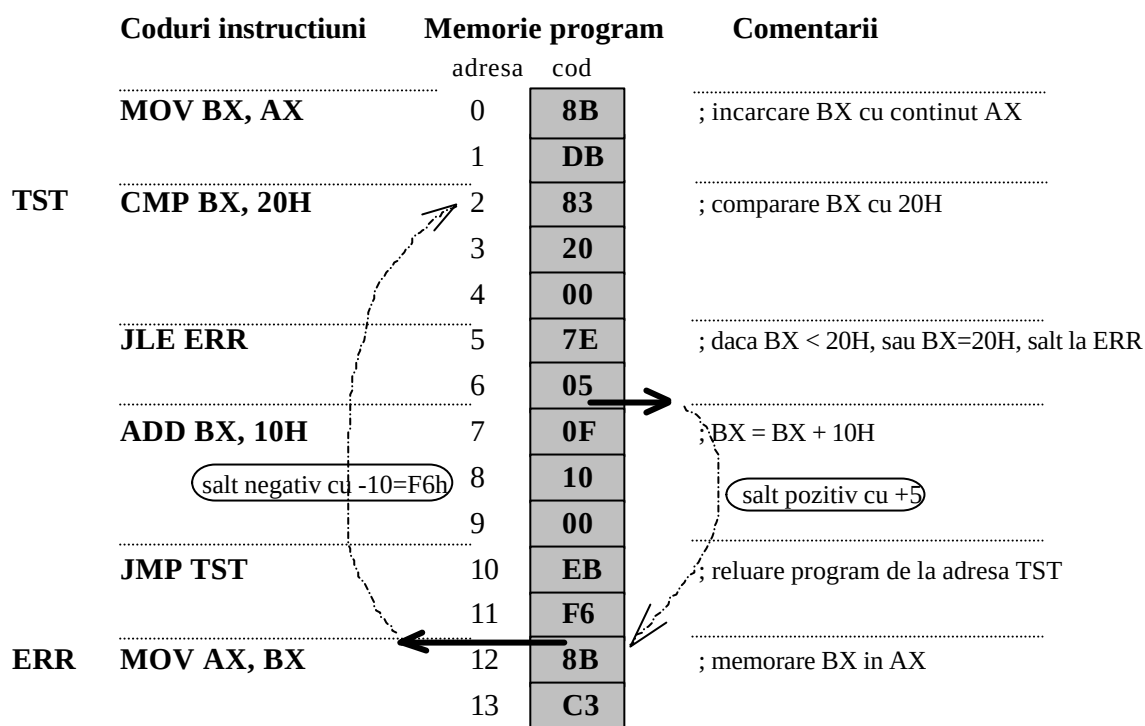


Fig. 3.43. Utilizarea adresarii relative la efectuarea salturilor scurte in program

3.5.3 Setul de instructiuni al microprocesorului

Gradul de complexitate si varietate al operatiilor ce pot fi efectuate de catre microprocesor definesc puterea de calcul si performantele posibile ale acestuia. Totalitatea operatiilor posibile: *intre registre, a celor cu memoria, aritmetico-logice si alte operatii speciale*, prin care se activeaza diversele parti constitutive ale microprocesorului (prezentate la sectiunea de *hardware*) alcatuiesc, in ansamblul lor, **setul de instructiuni** al microprocesorului. El permite "conectarea" intre ele a diverselor parti ale microprocesorului, dand flexibilitatea deosebita a acestuia. O simpla modificare de program a uneia sau a mai multor instructiuni, si functionarea globala a sistemului este alta. Iata un nou mod de "construire" a unui sistem automat.

Pentru a prezenta, in *mod principial*, modul de organizare al setului de instructiuni al unui microprocesor, se considera un procesor organizat pe cuvinte de 8 biti. Cele 256 de combinatii posibile din codul operatiei vor fi suficiente pentru a genera un set de instructiuni tipic.

Clasificarea instructiunilor

- **Instructiuni de transfer si actualizare a informatiei**

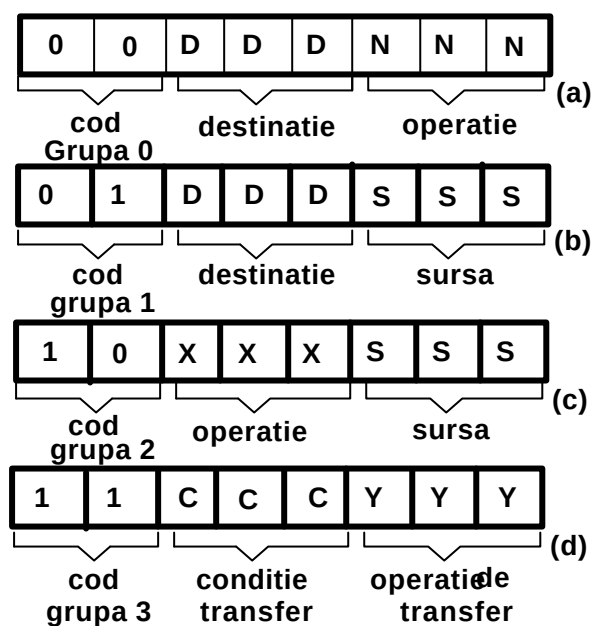


Fig.3.44. Structura codurilor operatiilor unui microprocesor

Grupul 0 --- operatii generale. Instructiunile acestui tip permit incrementarea, decrementarea (cresterea, respectiv scaderea continutului registrelor, sau a unei celule de memorie), incarcarea continutului registrelor, rotirea continutului acestora si alte operatii de acest gen.

Grupul 1 --- operatii de transfer al informatiei. Contine instructiunile ce permit transferarea informatiei dintr-o locatie denumita sursa, intr-o alta locatie, destinatia, fara a se modifica informatia. Sursele pot fi registre, celule de memorie RAM, PROM, porturi de intrare. Destinatiile pot fi registre, celule de memorie RAM, porturi de iesire. De facto, aceste operatii realizeaza legatura intre o locatie si alta (ca si cum ar fi conectate fizic una cu

alta).

- **Instructiuni de modificare a informatiei**

Grupul 2 --- operatiile aritmetico-logice. Contine instructiunile ce permit efectuarea de operatii aritmetice sau logice, asupra continutului unor registre ale microprocesorului, sau (doar pentru unele microprocesoare), asupra unor celule de memorie.

- **Instructiuni de control**

Grupul 3a --- operatii de control al programului. Sunt instructiunile care permit transferarea executiei programului, de la locatie curenta, intr-o alta zona a programului (modificarea PC). Transferul se poate efectua neconditionat, sau doar in cazul setarii unora dintre indicatorii registrului de stare. In plus, transferurile pot fi cu revenire in punctul de unde se genereaza transferul, in care caz adresa curenta a programului este salvata inainte de a se efectua transferul, sau fara revenire, caz in care nu se salveaza adresa curenta.

Grupul 3b --- operatii de control al procesorului. Sunt operatii ce actioneaza direct asupra microprocesorului. Tipice sunt operatiile de stop al procesorului, activare / dezactivare intreruperi, instructiunea inoperanta (NOP - no operation).

Se utilizeaza unii dintre bitii octetului de cod al instructiunii pentru a desemna grupa de operatii din care face parte instructiunea. Astfel, pentru cele patru grupe anterior definite, vor fi necesari 2 biti (fie cei mai semnificativi ai octetului), care definesc grupa careia ii apartine o operatie data.

În figura 3.44 sunt prezentate structurile codurilor operațiilor instrucțiunilor din cele patru grupe, pentru un microprocesor de 8 biți (pentru simplitatea prezentării).

Astfel, **instrucțiunile din grupa 0** sunt formate din cei doi biți de cod de grupă, trei biți (desemnați prin **DDD**), codificând registrul asupra căruia se operează (se remarcă deci limitarea la 8 registre --- sau, uzual, 7 registre și o celulă de memorie, adresată indexat de către un registru index) și trei biți (**NNN**), indicând operația ce se efectuează. Cunoșcând, din manualul de programare al microprocesorului, codurile fiecărui registru al acestuia, cât și ale operațiilor dorite a se efectua, se poate genera codul unei anume instrucțiuni din această grupă. De exemplu, considerând o operație de incrementare (cod **100**) a registrului 3, (cod **011**), codul binar al operației va fi (**DDD=011**, **NNN=100**), **00011100** sau, într-o formă accesibilă, în baza 8 de numărare, **034g**.

Scrierea unor programe sub această formă, (desemnata ca **programare în cod masina**), ar fi nu numai greoaie, dar și expusă la foarte probabile erori. De aceea, se utilizează așa-numitele **mnemonici**, cuvinte abreviate, de proveniență engleză, semnificând operația ce se efectuează, sursa și destinația ei (unde este cazul). Ca un exemplu, instrucțiunea anterioară poate fi referită ca **INC R3**; (**incrementează R3**).

Trebuie accentuat că scrierea programelor cu aceste *mnemonici*, alături de ceea ce se numește **limbajul de asamblare** al microprocesorului (și fiind specific fiecărui procesor în parte), permite elaborarea programelor "pe hârtie". Fizic, procesorului i se va pregăti în memorie succesiunea de *coduri binare*, corespunzătoare acestor *mnemonici*, conform *setului de instrucțiuni*, care vor corespunde programului în limbaj de asamblare. Operația de trecere de la limbajele de asamblare la programul executabil poate fi făcută sau prin elaborarea de către programator a codurilor corespunzătoare instrucțiunilor și introducerea lor (de exemplu) de la niste comutatoare, în memoria sistemului (programarea fiind practic efectuată în cod masina), sau prin executarea, la sistemele mai evoluate, a unui program special, numit **asamblor**. Acesta poate *prelua* textul, denumit **text sursă**, continuând programul scris în *limbaj de asamblare* (de exemplu prin citirea de la un periferic a unui fisier), *analizează* aceste instrucțiuni, *generând*, în cazul corectitudinii lor, **codul binar** corespunzător operațiilor dorite. (Evident, trebuie date indicații suplimentare, referitoare la adresa de memorie unde se va genera codul programului etc., probleme ce nu fac obiectul acestui paragraf).

Asamblorul reprezintă o prima etapă în gradul de "evoluție" a limbajelor în care poate fi programat un microprocesor. El realizează o corespondență "unu la unu", fiecare instrucțiune în limbaj de asamblare generând cod pentru o singură instrucțiune a microprocesorului. Totodată, limbajul de asamblare este propriu fiecărui tip de microprocesor. Există **limbaje de nivel înalt**, care permit programarea microprocesorului cu instrucțiuni puternice (limbaj *FORTRAN*, *PL/M*, *BASIC*, *PASCAL*, *C*, *C++*, *ADA*, *FORTH*. etc.). Dispunând de programele specifice, numite **compilatoare** (specifice pentru fiecare limbaj), se pot genera, din *texte sursă*, scrise în aceste limbaje, *coduri obiect* pentru microprocesorul respectiv. Aceste programe vor necesita pentru a fi executate, sisteme de calcul, uzual așa-numitele **sisteme de dezvoltare**, microcalculatoare de putere sporită, cu facilități specifice în ceea ce privește resursele sistemului (memorie, suporturi de memorie externă de tip disc, etc.).

Instructiunile grupei 1 au structura conform figurii 3.44b. Cele doua grupe de cate trei biti, *DDD* si *SSS* semnifica *registrul destinatie*, respectiv *sursa*, pentru efectuarea transferului. Transferul informatiei din registrul *R3* (cod 011) in *R6* (cod 110) va avea deci codul **163g**. ({1} inseamna cod operatie de transfer, {6} cod registru destinatie, {3} cod registru sursa). Mnemonica corespunzatoare poate fi scrisa ca **MOV R6,R3**.

Pentru **operatiile aritmetico-logice**, codul operatiei este dat in figura 3.44c. Aici, grupul *XXX* de trei biti indica operatia ce se efectueaza, iar *SSS*, registrul *sursa*, continand unul dintre operanzi. In mod implicit, celalalt operand este considerat a fi continut de catre *acumulator*. Totodata, rezultatul este memorat tot in *acumulator* (alterand operandul continut de catre acesta). Astfel, o instructiune **ADD R4**, va aduna la continutul *acumulatorului*, continutul registrului *R4*, rezultatul fiind plasat in *acumulator*.

Instructiunile grupei a treia nu pot fi clasificate intr-o maniera la fel de clara ca pentru grupele anterioare. Totusi, o pondere speciala o au, in aceasta grupa, **instructiunile de ramificare** a programului. Acestea permit, dupa cum s-a mentionat, devierea executiei programului de la secventa normala de instructiuni, operatie necesara in anumite puncte de decizie (in functie de un anume rezultat, sau data de intrare, de exemplu, se va initia o anume actiune sau alta) sau, in mod neconditionat, in unele puncte din program.

Ramificarea conditionata se bazeaza pe examinarea registrului de stare, care contine informatii referitoare la rezultatele ultimei operatii aritmetice sau logice efectuate in ALU. Dupa cum s-a mentionat, indicatorii de stare continuti in acest registru pot specifica:

- rezultat nul (*Z --- zero*), sau nenul (*NZ --- not zero*);
- a aparut depasire (sau imprumut) (*C --- carry*), sau nu (*NC --- not carry*);
- rezultatul are un numar par de biti 1 (*PE -- parity even*), sau impar (*PO -- parity odd*);
- rezultat pozitiv (*P --- plus*), sau negativ (*M --- minus*); etc.

Ca exemple tipice de instructiuni de ramificare conditionate, vom mentiona instructiunile **JUMP**, **CALL** si **RETURN** (toate putand fi instructiuni conditionate sau nu).

Instructiunea **JUMP ("salt")**, permite saltul programului la o noua adresa, in cazul indeplinirii unei conditii in *registrul de stare*. De exemplu, saltul la aparitia unui rezultat nul, simbolizat prin mnemonica **JZ**, urmata de adresa dorita la care se comuta programul. Daca indicatorul testat nu este setat, (de exemplu, nu a fost rezultat nul), se continua executia programului, cu instructiunea urmand celei de salt conditionat (nu se efectueaza saltul).

Instructiunea **CALL ("cheama")** permite apelarea de *subrutine*. Acestea reprezinta zone de program (denumite si *subprograme*), care se vor executa de mai multe ori in decursul operarii sistemului. Aceste subprograme pot fi "chemate" ("apelate") din programul "principal", sau din alte subrutine, prin instructiuni de tipul **CALL**. Transferul executiei programului la adresa unde se gaseste subrutina poate fi deasemenea conditionat si se executa din acest punct de vedere ca si pentru instructiunile de salt. Ceea ce deosebeste acest transfer de instructiunile de salt, il reprezinta *salvarea*

continutului *PC*, din punctul de unde se face apelul. Aceasta permite ca, la terminarea subrutinei apelate, prin executarea unei instructiuni de tip **RETURN** ("*intoarcere*"), sa se refaca in *PC* valoarea salvata la apelare. Se determina astfel continuarea programului apelant, de la instructiunea imediat urmatoare celei de apelare (*CALL*).

Si instructiunea de **RETURN** poate fi conditionata. Salvarea (refacerea) continutului *PC* este facuta cu ajutorul *registrului indicator de stiva (SP)*. La executarea unei instructiuni de salt la subrutina, microprocesorul va depune in memoria sistemului, la adresa indicata de *SP*, valoarea numaratorului de program, *PC*. Aceasta *zona de memorie RAM*, unde se depune valoarea *PC* la apelare de subrutine si unde se mai poate depune si continutul registrelor microprocesorului (prin instructiuni specifice de salvare a acestora, necesare la tratarea intreruperilor), se numeste **stiva** programului. Organizarea acesteia in memoria externa microprocesorului permite realizarea de stive oricat de mari. Dupa fiecare salvare in stiva, *SP* se modifica, continand adresa varfului noii stive (permitand salvari succesive, nealterand vechile valori din stiva). Aceasta permite executarea de apelari de subrutine, din alte subrutine, inlantuit. La executarea instructiunilor de **RETURN**, se aduce, de la adresa indicata de *SP*, (din varful stivei), valoarea *PC* care fusese salvata la apelare si se reactualizeaza *SP*, micșorand stiva corespunzator. O asemenea inlantuire a apelurilor de subrutine este prezentata in figura 3.45.

Evident, este raspunderea programatorului de a respecta ordinea strict inversa celei de la apelare, in cazul unor inlantuiiri de apeluri (ordine *LIFO* --- *last in, first out* --- *ultimul intrat, primul iese*). Deasemenea, daca s-au salvat registre in stiva, ele trebuiesc refacute etc.

Pentru *stiva*, se rezerva o anumita zona din memoria sistemului. Nu exista uzual o marime standard a stivei, limitarile fiind legate de disponibilul total de memorie, aflat la dispozitia programatorului pentru a-l aloca stivei. *Principalul avantaj* al utilizarii stivei este ca permite memorarea temporara a datelor fara complicatia de a specifica in mod explicit adresa cu care se opereaza (necesara la adresarile tipice ale memoriei - vezi modurile de adresare). Utilizatorul nu cunoaste si nici nu trebuie sa cunoasca adresa unde se memoreaza datele, operatiile desfasurandu-se indirect, prin utilizarea *registrului indicator de stiva*. Principala limitare a acestei modalitati de stocare a datelor este data de modul de acces la stiva, secvential. Detalii privind utilizarea stivei vor fi date ulterior, intr-un alt paragraf.

In afara instructiunilor de ramificare a programului, mai exista **operatii speciale**, cu *registrul indicator de stiva*, *operatii de intrare/iesire*, *operatii de control al microprocesorului* etc., pentru fiecare tip de procesor in parte.

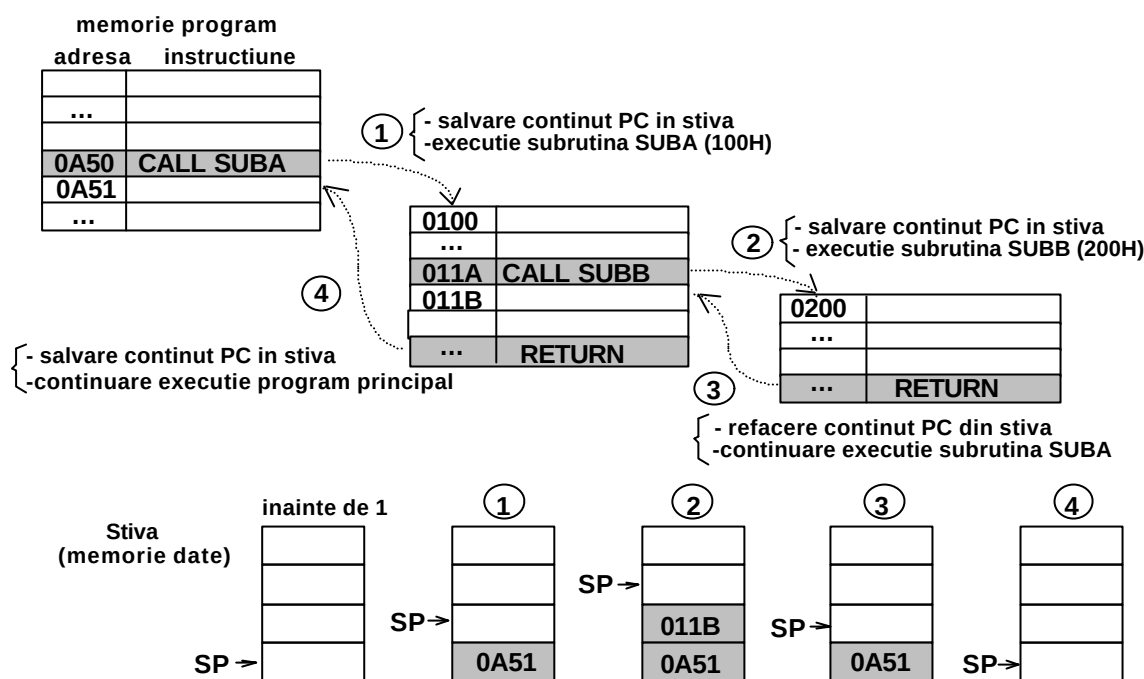


Fig.3.45. Modul de utilizare a stivei la apelarea subrutinelor

Pentru a sesiza mai bine corelarea ansamblului de operatii ce se efectueaza pentru executarea unei instructiuni a microprocesorului, se prezinta, in figura 3.46, respectiv 3.47, in mod schematic, transferul de informatie si operatiile efectuate in fiecare perioada de ceas in parte, pentru doua instructiuni ale unui microprocesor de 8 biti (ZILOG 80).

Astfel, figura 3.46 prezinta executia instructiunii **ADD R** de adunare la continutul acumulatorului, a continutului registrului **R** al microprocesorului. Se remarca executia instructiunii intr-un singur subciclu, de patru perioade de ceas. Primele trei perioade de ceas se folosesc pentru extragerea si decodificarea instructiunii, iar in al patrulea se executa operatia. Durata acestei instructiuni va fi deci de $4 \times T_c$. (Pentru un ceas de 2 MHz, instructiunea va dura deci 2 microsecunde).

Instructiunea **LDA adr** (cel de-al doilea exemplu), implica incarcarea continutului acumulatorului cu continutul celulei de memorie de la adresa **adr**. Figura 3.47 prezinta desfasurarea acestei instructiuni, pentru care se vor efectua 4 accese la memorie (3 pentru citirea codului instructiunii - 1 cod operatie, 2 operanzii, adresa de unde se face citirea - adresare directa, si unul pentru citirea datei dorite, de la adresa **adr**). Se remarca aparitia a 4 subcicluri pentru executarea acestei instructiuni, durand 13 perioade de ceas (deci 6,5 microsecunde).

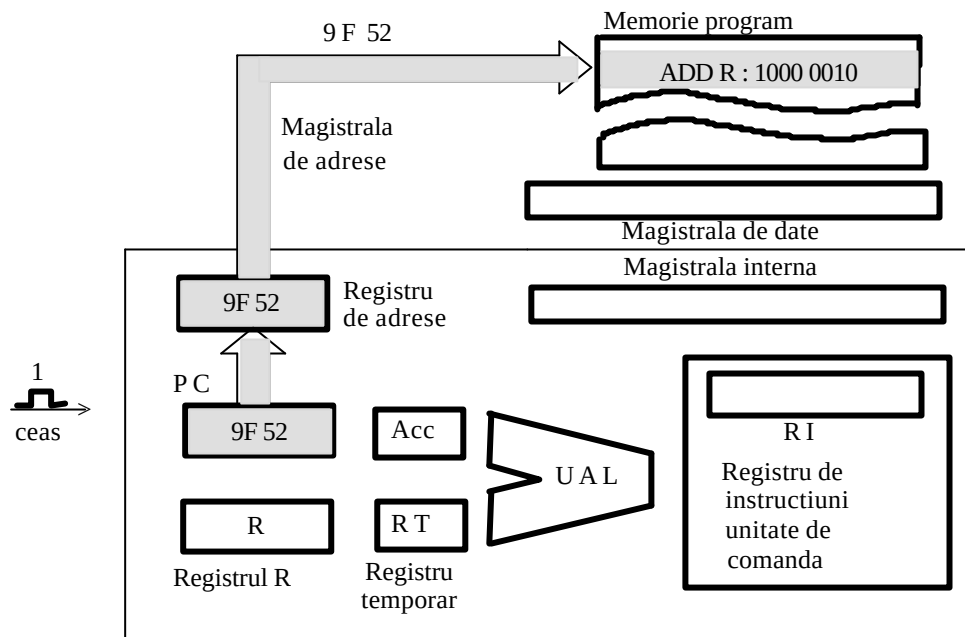


Fig.3.46. Executia instructiunii ADD R. Registrul contor de program are valoarea 9F52_H
(a)(Subciclul 1, T1) Se selecteaza celula de memorie program la adresa 9F52_H
(valoarea curenta a PC)

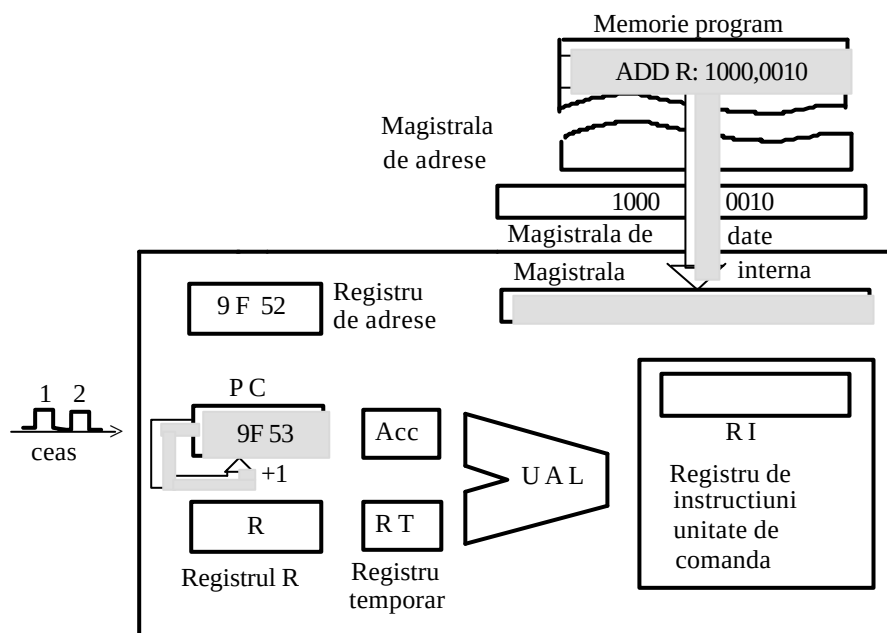


Fig.3.46.b. (Subciclul 1, T2) Codul instructiunii (din celula de memorie selectata) pe magistrala de date; se incrementeaza PC

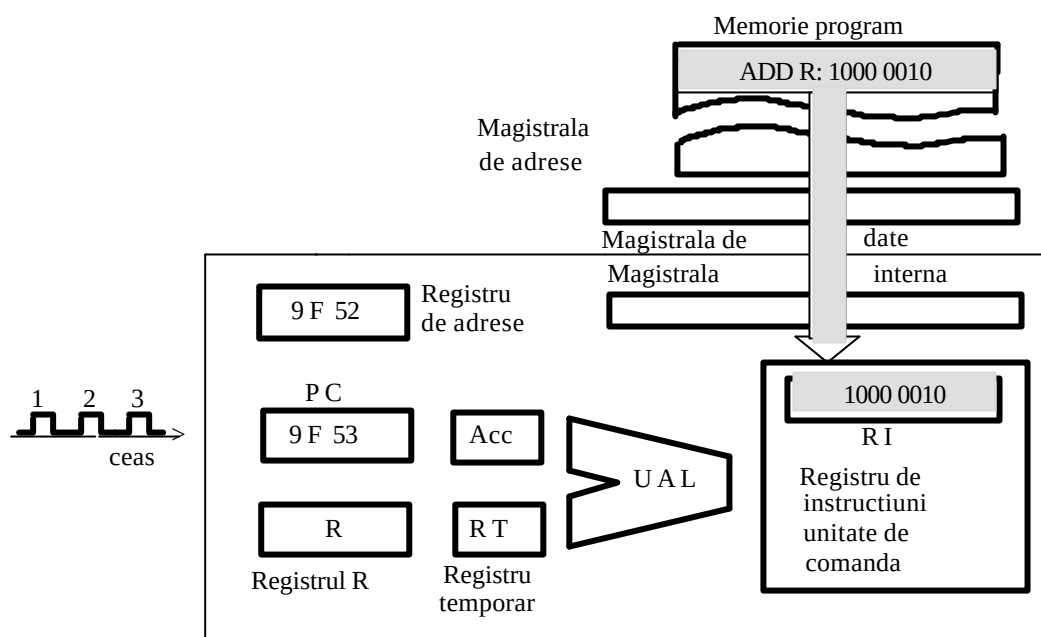


Fig.3.46.c. (Subciclul 1, T3) Codul instructiunii citit pe magistrala de date, in registru de instructiuni

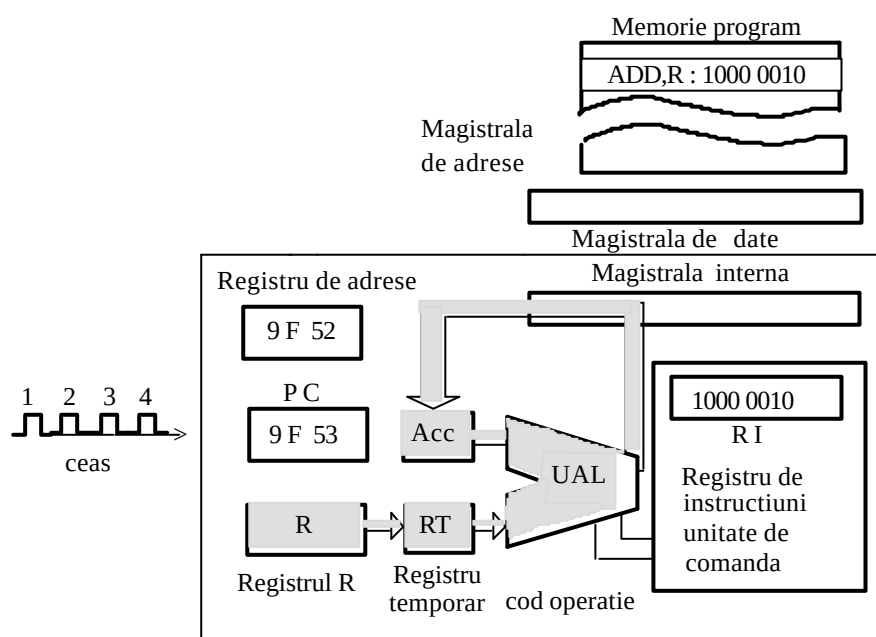


Fig.3.46.d. (Subciclul 1, T4) Se decodifica si executa instructiunea; se aduna continutul registrului R la acumulator; rezultatul in acumulator

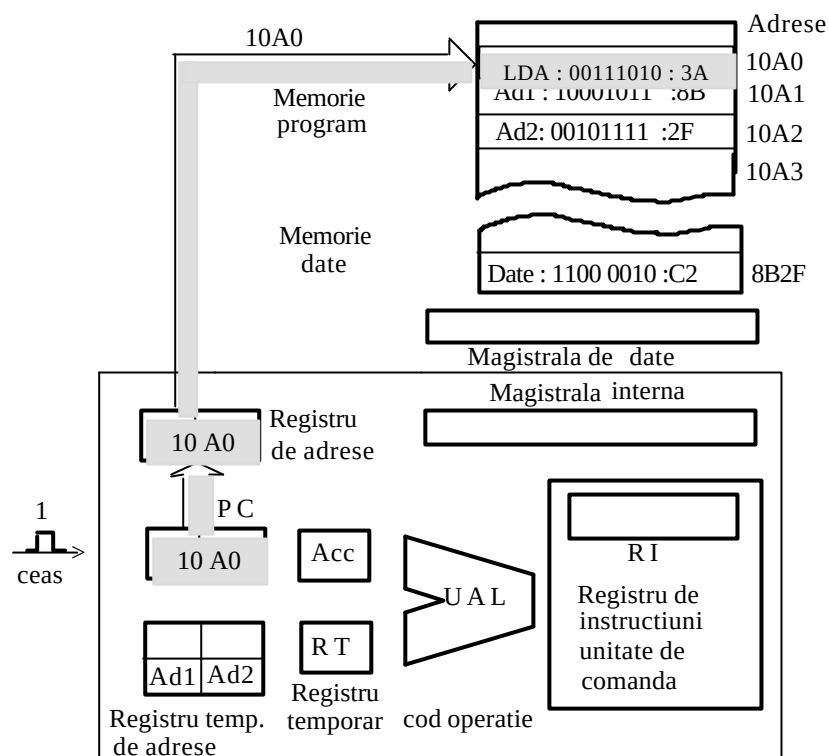


Fig.3.47. Instrucțiunea de incarcare a acumulatorului de la adresa ADR = 8B2FH. continuta in instructiune, PC are valoarea 10A0H. Subciclul 1 -- se va citi si decodifica ciclul instructiunii:

(a) T1. Se selecteaza celula de memorie program de la adresa 10A0H

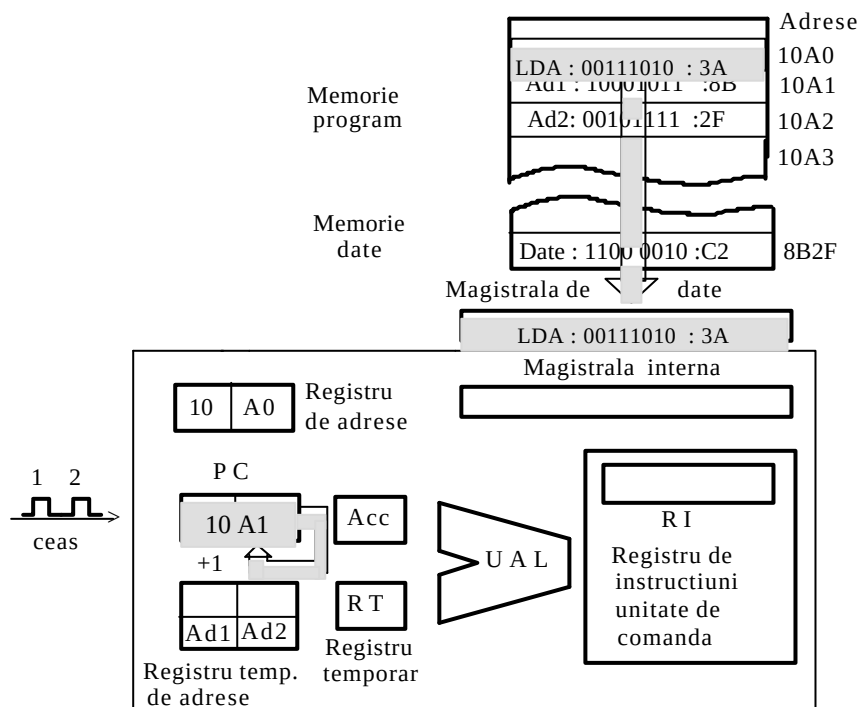


Fig.3.47.b. T2. Codul instructiunii pe magistrala de date; se incrementeaza PC

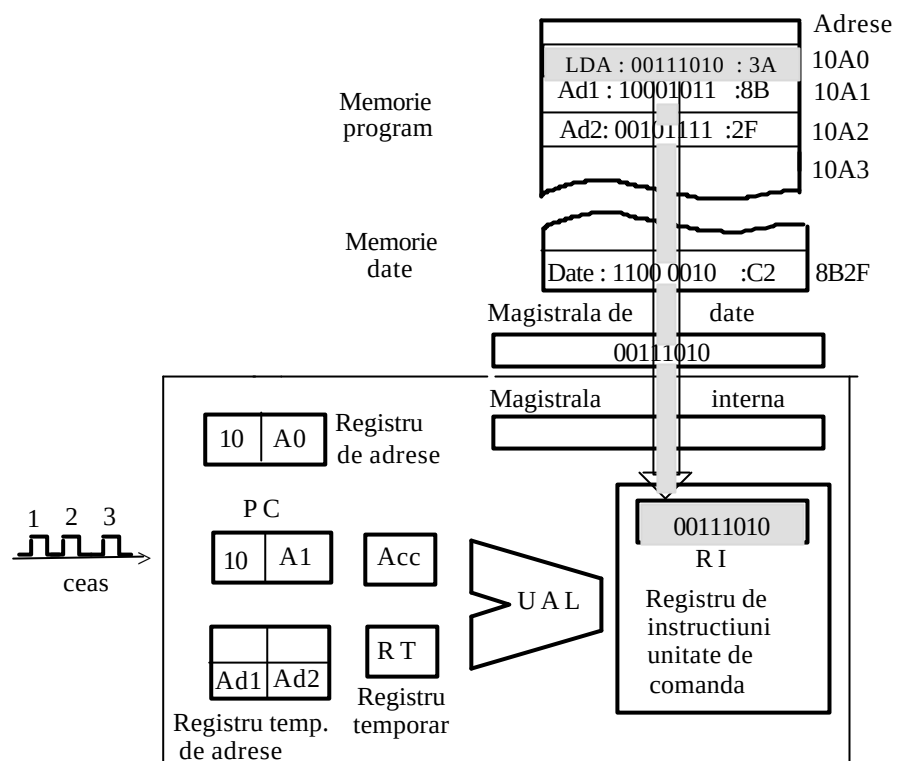


Fig.3.47.c. T3. Codul instructiunii citit de pe magistrala de date in registru de instructiuni

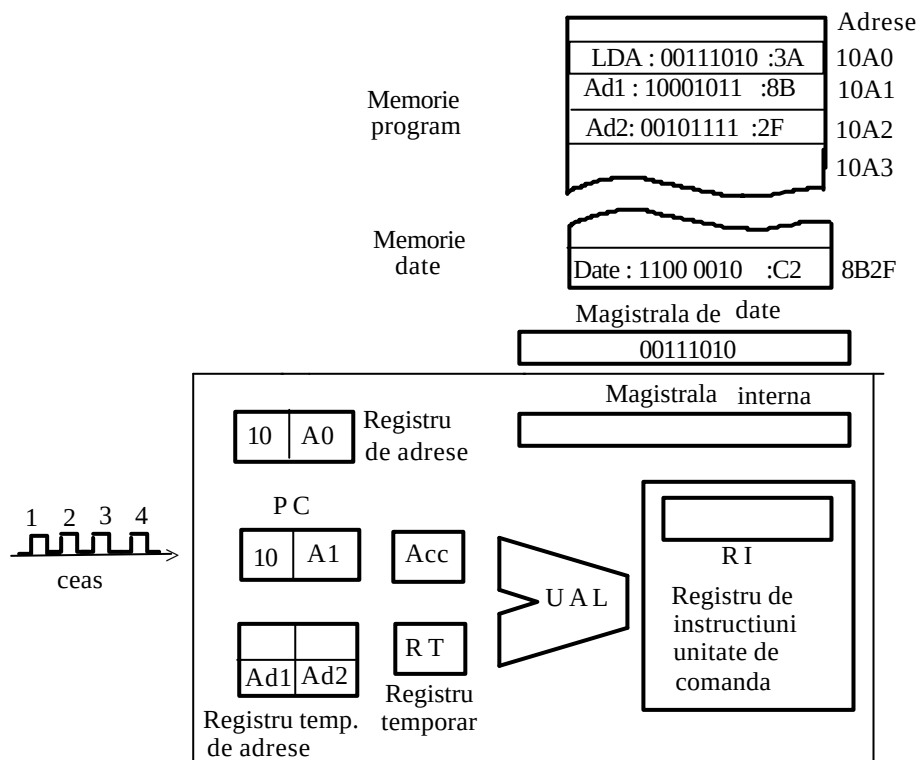
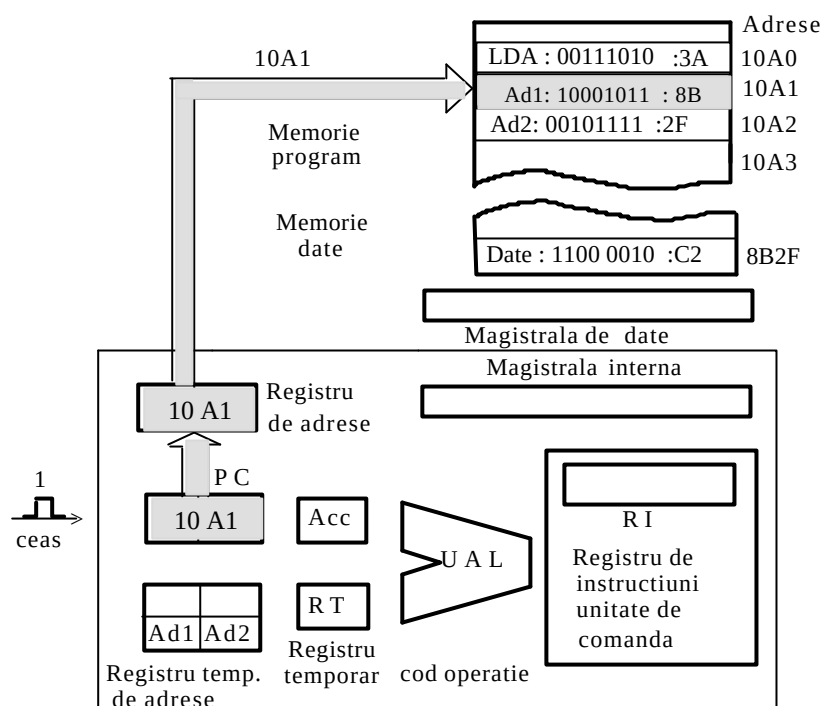


Fig.3.47.d. T4. Se decodifica codul instructiunii



Subciclul 2 -- se va citi cel de-al doilea octet al instructiunii (data)
 Fig.3.47.e. T1. Se selecteaza celula de memorie program de la adresa 10A1; celula
 contine cei 8 biti superiori ai adresei ADR

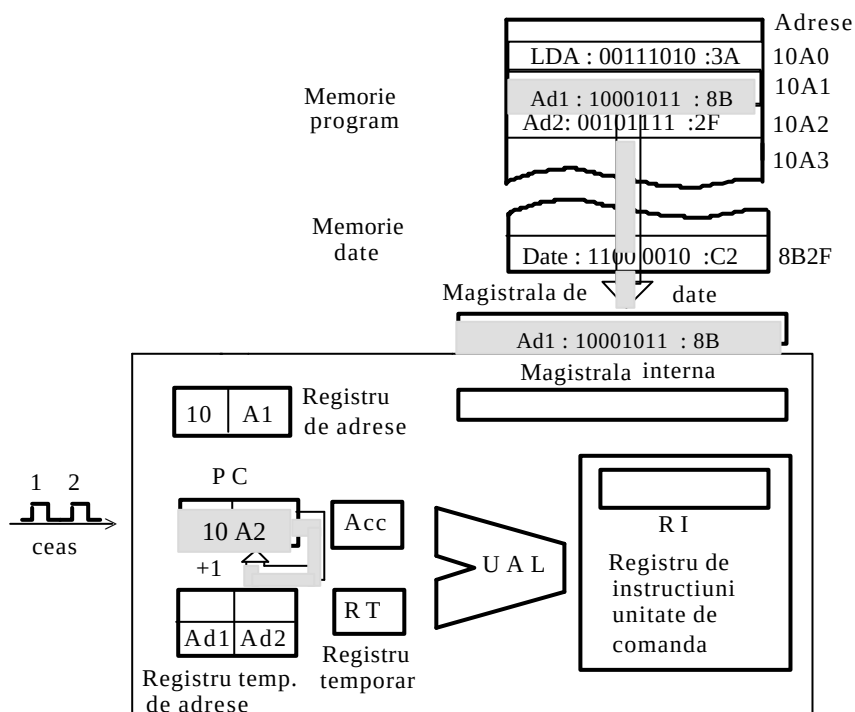


Fig.3.47.f. T2. Valoarea octetului cel mai semnificativ al adresei ADR pe magistrala
 de date; se incrementeaza PC

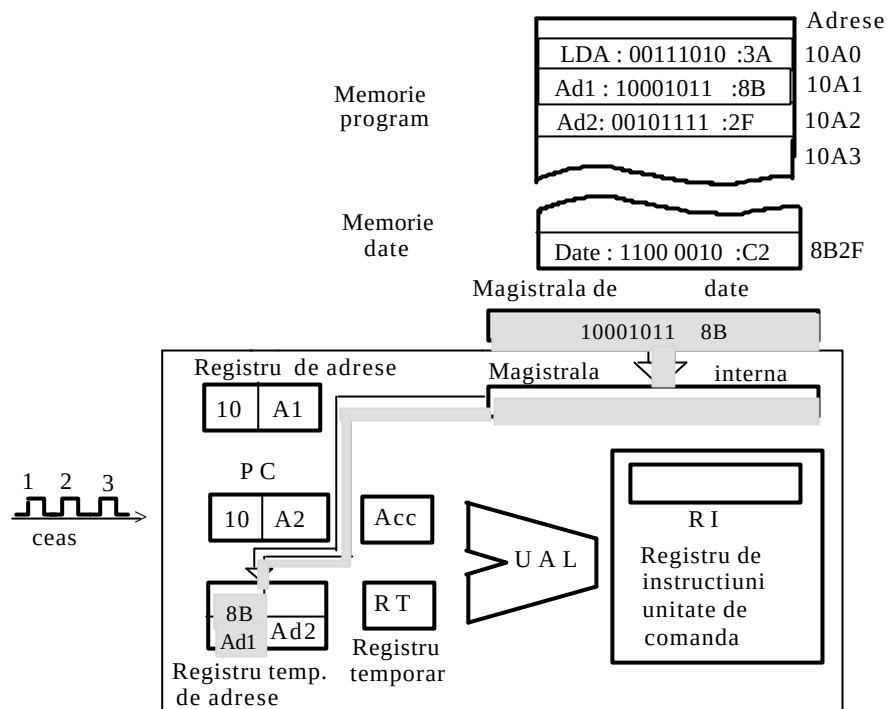
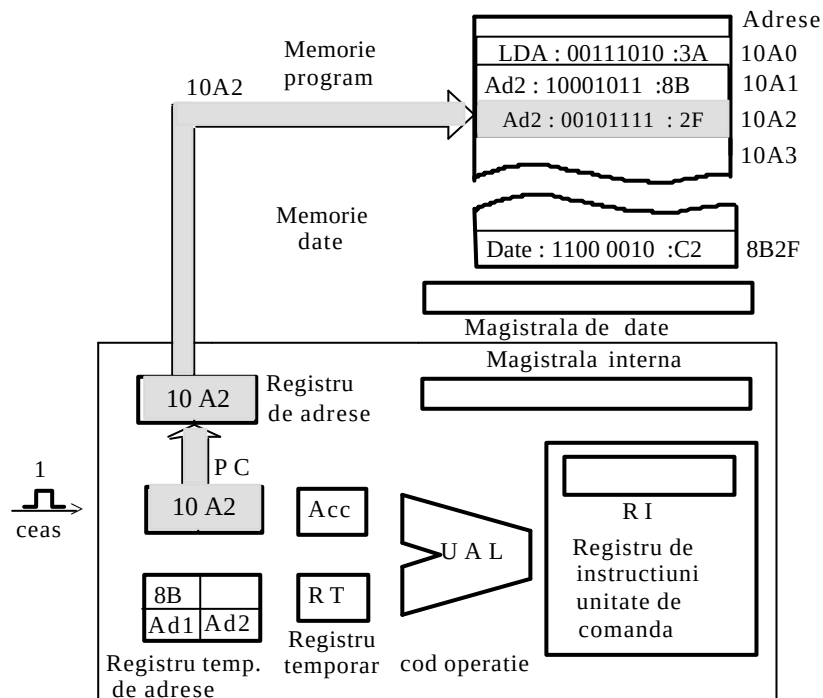


Fig.3.47.g. T3. Memorie program and date, and internal bus. The diagram shows the state of the processor at time T3. The program memory contains LDA : 00111010 :3A at 10A0, Ad1 : 10001011 :8B at 10A1, and Ad2: 00101111 :2F at 10A2. The data memory contains Date : 1100 0010 :C2 at 8B2F. The internal bus is 10001011 8B. The processor registers are: PC (10 A2), Acc, RT (8B Ad1), and RI (10 A1). The ALU is labeled U A L. The instruction register is labeled RI. The address register is labeled 10 A1. The data register is labeled 8B Ad1. The internal bus is labeled Magistrala de date and Magistrala interna. A clock signal 'ceas' is shown with three pulses labeled 1, 2, and 3.



Subciclul 3 -- se va citi al treilea octet al instructiunii (data)
Fig.3.47.h. T1. Se selecteaza celula de memorie program de la adresa 10A2_H (PC)
celula contine cei 8 biti inferiori ce compun adresa ADR

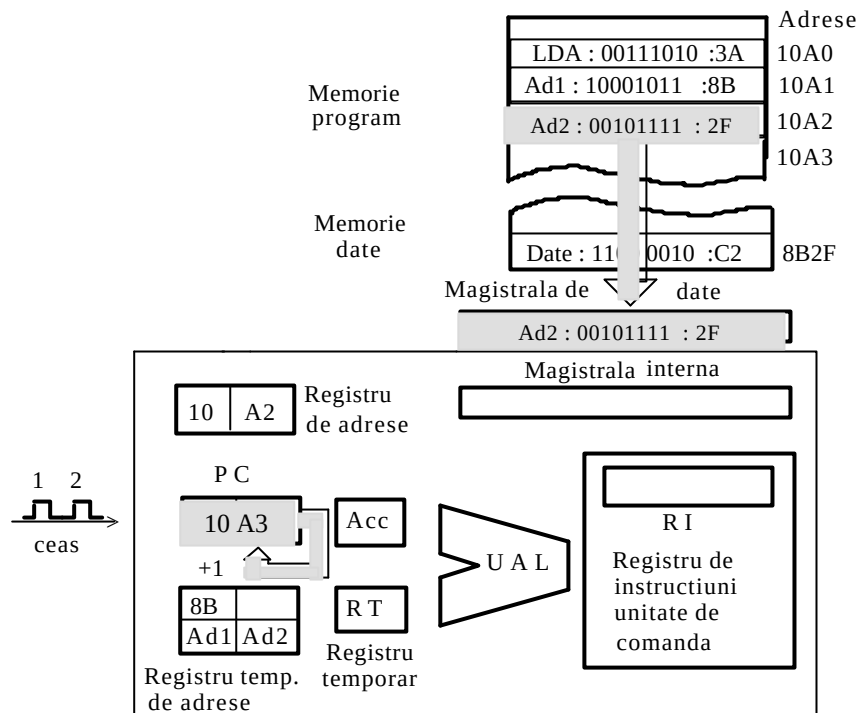


Fig.3.47.i. T2. Valoarea octetului cel mai puțin semnificativ al adresei ADR pe magistrala de date; se incrementează PC

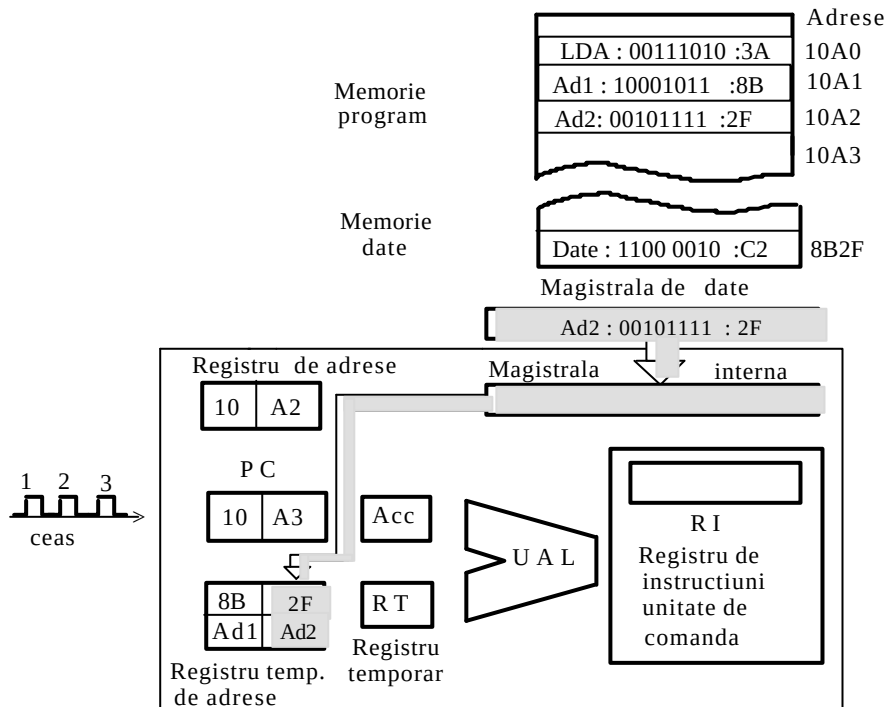
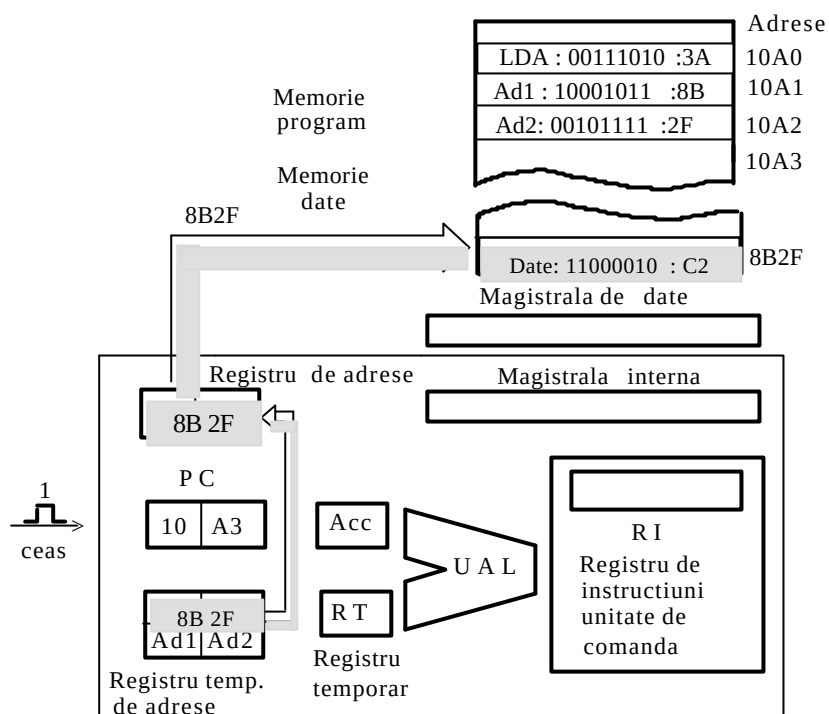


Fig.3.47.j. T3. Memorarea octetului cel mai puțin semnificativ al adresei ADR în partea superioară a registrului de adrese



Subciclul 4 -- se va executa efectiv instructiunea: se incarca acumulatorul cu valoarea de la adresa AD1

Fig.3.47.k. T1. Se selecteaza celula de memorie de date de la adresa 8B2F_H aflata in registrul tampon de adrese

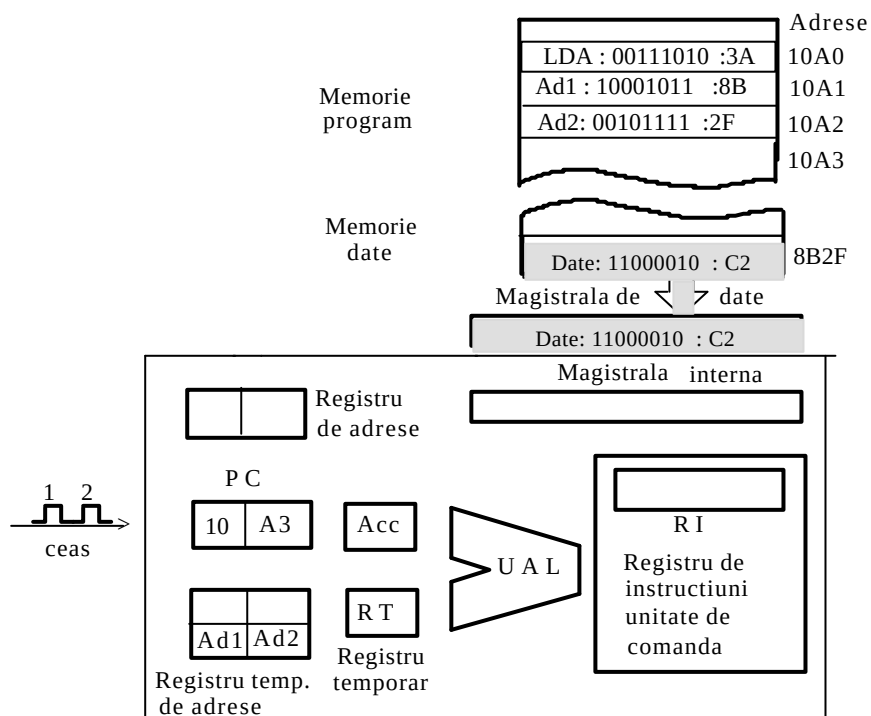


Fig.3.47.l. T2. Data din celula selectata pe magistrala de date

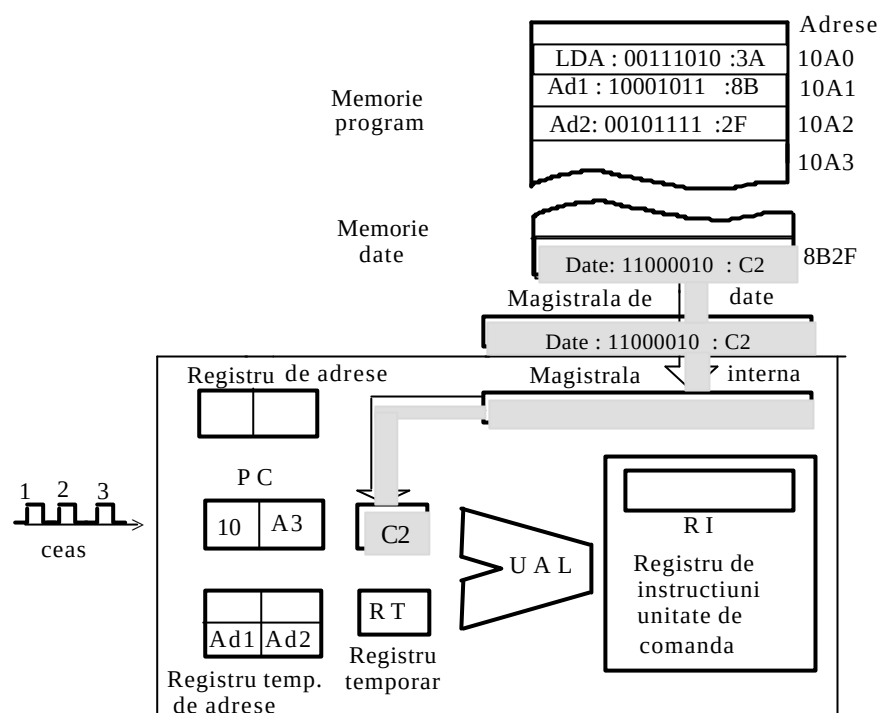


Fig.3.47.m. T3. Data din celula adresata este memorata in acumulator

3.5.4 Utilizarea stivei programului

Deși s-au făcut și în paragrafele precedente referiri și s-au dat unele explicații referitor la definirea și modul principal de utilizare a stivei, data fiind importantă conceptului, se vor da în prezentul paragraf detalii suplimentare în acest sens.

În mod implicit, nu există posibilitatea, în cursul executiei programului, de a se "rememora" adresa unei anumite instrucțiuni, după executia acesteia. Astfel, la apelarea unei subrutine, programatorul trebuie să indice la sfârșitul acesteia unde trebuie să se "reîntoarce" registrul indicator de program, pentru a continua executia normală a acestuia. Deoarece o subrutină poate fi apelată din puncte diferite ale programului, nu se poate face acest lucru prin revenire într-un punct fix al programului (a cărui adresă ar putea fi, eventual, indicată la revenirea din subrutină). Din cauza structurii explicite a instrucțiunilor, nu există această posibilitate în programele scrise în limbajul de asamblare; de aceea, programatorul trebuie să salveze adresa de revenire, ori de câte ori executia programului se transferă dintr-un punct (unde este memorat secvențial), în alta zonă a memoriei, unde este memorată alta parte a programului. Conceptul de stivă, introdus în paragrafele precedente, va fi utilizat pentru a explica elementele esențiale legate de operarea cu stivă, particularizat pentru apelarea subrutinelor (CALL), și revenirea din subrutină (RETURN).

După cum s-a menționat, stivă este organizată pe sistemul FIFO, cu registrul indicator de stivă conținând adresa următoarei locații libere din stivă. Se utilizează doi termeni legați de operațiile cu

stiva: salvarea in stiva (push), si refacerea din stiva (pop). Primul implica transferarea unei date in varful stivei, iar cel de-al doilea, extragerea unei date din varful stivei. Data respectiva poate reprezenta in sens general atat o adresa, necesara la revenirea dintr-o subrutina, cat si argumente (valori, sau continutul unora dintre registrele microprocesorului) transmise prin stiva, subrutinei, sau returnate programului apelant. Figura 3.48 prezinta acest mecanism de utilizare a stivei in cele doua scopuri.

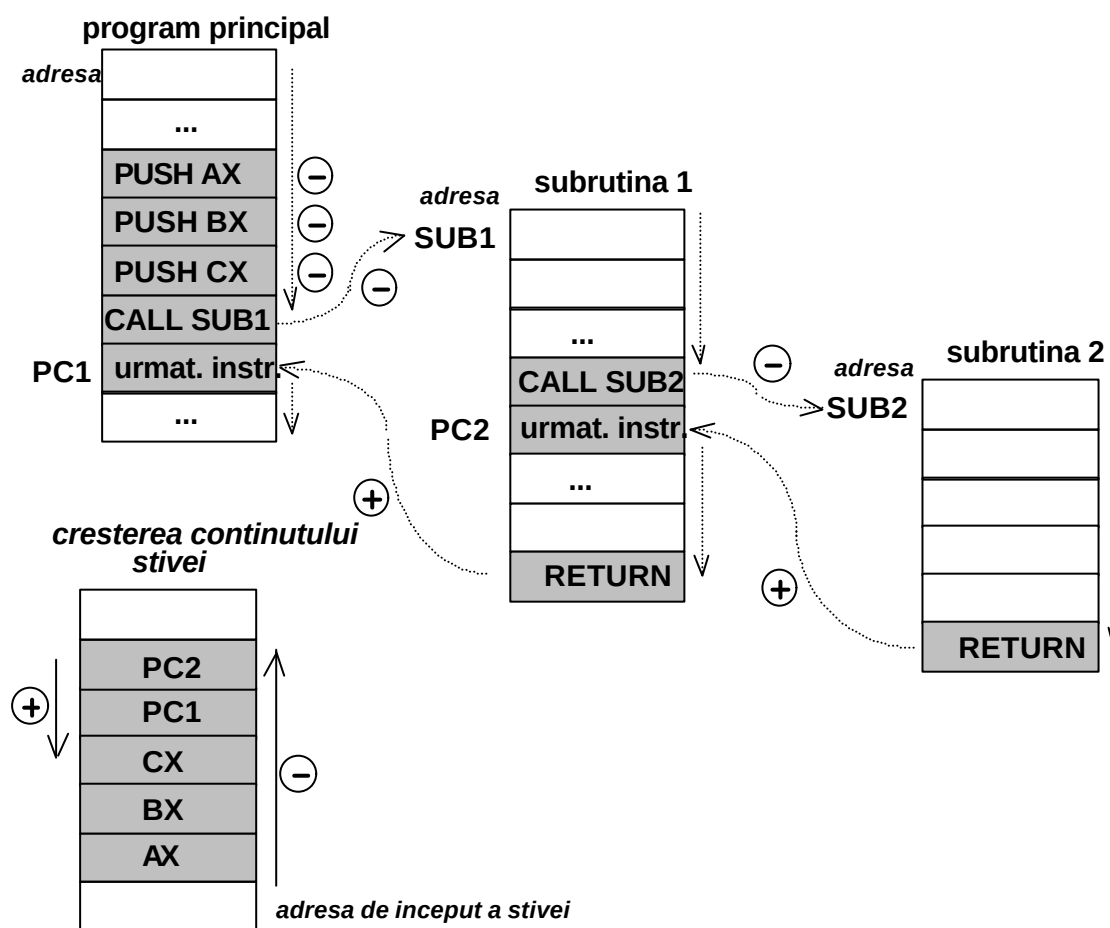


Fig. 3.48. Salvarea continutului registrelor si a adreselor de revenire in stiva

Programatorul trebuie sa tina seama de urmatoarele elemente relativ la utilizarea stivei:

- stiva reprezinta o zona de memorie, operand in modul LIFO;
- orice cuvint de date salvat in stiva, va trebui extras de acolo la un moment dat in cursul executiei programului;
- o consecinta a modului LIFO de lucru cu stiva este ca informatia poate fi refacuta din stiva in ordine inversata fata de cea de la salvarea in stiva;

- creșterea stivei este corelată cu scăderea conținutului registrului indicator de stivă; astfel, când stiva este goală, SP are valoarea inițială (maximă); când stiva este plină, SP are o valoare minimă. Stiva crește către baza (valorile mici) ale memoriei sistemului.

Apelarea succesivă a unei subrutine din alta subrutină, în lanț, este posibilă în principiu fără alte restricții decât memoria aflată la dispoziție, alocată stivei. Mai mult, este posibilă chiar apelarea unei subrutine de către ea însăși (apel recursiv), utilizată uzual corelat cu testarea unui indicator care asigură ieșirea din buclă recursivă.

Subrutinele și stiva

Există trei operații esențiale, asociate execuției corecte a unei subrutine:

- salvarea registrului indicator de program (PC), reprezentând adresa instrucțiunii imediat următoare celei de apelare a subrutinei (prima instrucțiune de după cea de CALL); după execuția subrutinei, execuția programului se reia de la această instrucțiune;
- executarea subrutinei apelate;
- revenirea la adresa memorată la apelare, și continuarea execuției din punctul respectiv.

Aceste operații se efectuează utilizând cele două operații complementare, de apelare a subrutinei (CALL) și, respectiv, de revenire din subrutină (RETURN). Trebuie remarcat că la orice operație de salvare în stivă, după salvarea valorii respective (PC la apelarea unei subrutine), conținutul registrului indicator de stivă SP este actualizat automat (decrementat), permițând o eventuală salvare ulterioară, de exemplu în cazul unui alt apel, în lanț, al unei alte subrutine. La orice operație de revenire din subrutină, se actualizează SP, (prin incrementare), și doar apoi se reface conținutul PC, cu valoarea care se citește din stivă, reprezentând chiar valoarea salvată la apelarea subrutinei.

Transmiterea parametrilor

La apelarea sau revenirea dintr-o subrutină, este necesară comunicarea de date între subrutine, în afara de salvarea adresei de revenire. Acest proces de comunicare între subrutine este cunoscut sub numele de transmitere a parametrilor (argumente, sau variabile). Se pot evidenția trei metode de bază în transmiterea parametrilor între subrutine, ce vor fi descrise în continuare.

Prima și cea mai simplă metodă de transmitere a parametrilor este de a asocia fiecărei subrutine un set de locații de memorie, unde programatorul memorează date necesare a fi prelucrate de o altă subrutină, apelată de prima. Aceste locații de memorie pot conține atât date cât și adrese ale unor date (variabile). Principala limitare a acestei metode (simplă, în esență), este blocarea unor locații de memorie care vor fi utilizate doar pentru apelarea rutinei respective.

A doua metoda de transmitere a parametrilor se bazeaza pe utilizarea registrelor de uz general. Inainte de a apela o subrutina, se incarca aceste registre cu argumentele ce trebuiesc transferate subrutinei. Problemele asociate cu aceasta metoda (deasemenea simpla), sunt: necesitatea de a salva continutul registrelor, utilizate la apelarea rutinei, pentru reutilizarea continutului lor dupa revenirea din subrutina; limitarea introdusa de numarul relativ redus al registrelor de uz general.

A treia metoda de comunicatie este o forma speciala a celei anterioare, si se bazeaza pe utilizarea stivei pentru memorarea temporara a datelor ce trebuiesc transferate subrutinei. In acest scop, majoritatea microprocesoarelor au in setul de instructiuni operatii de transfer in/din stiva a continutului registrelor de uz general (PUSH, respectiv POP, la microprocesorul 8086), care permit salvarea/refacerea explicita in/din stiva, la executia instructiunilor respective, a registrelor procesorului. Argumentele ce trebuiesc transmise subrutinei pot fi astfel salvate in stiva, apoi se face apelul subrutinei; la intrarea in subrutina, se fac refaceri din stiva, citindu-se aceste valori, ce pot astfel fi utilizate in cursul executiei subrutinei. Operatii de acest gen necesita, dupa cum s-a mai mentionat, o mare atentie din partea programatorului, pentru a nu afecta procesul de revenire din subrutina (regasirea continutului PC salvat la apelarea rutinei, in momentul revenirii la programul apelant).

Registrul indicator de stiva trebuie initializat la lansarea programului, cu adresa de inceput a stivei. Uzual, incarcarea stivei se face de la adrese mari spre cele mici, deci la start, continutul registrului *SP* are o valoare uzual reprezentand limita superioara a memoriei de date. *Programului si variabilelor* programului li se alocata, tot uzual, locatiile inferioare ale memoriei de date. Pe cand aceste locatii sunt atribuite si fixate pe tot parcursul rularii programului (deci ocupa permanent o zona din memorie), stiva se modifica in mod *dinamic*, dupa cum s-a mai precizat. *Principala problema* in timpul utilizarii stivei este legata de *cresterea* acesteia la dimensiuni prea mari, conducand la *suprapunerea* acesteia *peste zona de date sau de cod de program*. Daca se ajunge la o asemenea situatie, (posibila in principiu, deoarece evolutia stivei poate scapa de sub estimarea programatorului), efectele pot fi catastrofale pentru executia programului, complet imprezibile (de exemplu, daca stiva patrunde in zona de cod de program, acesta va fi alterat, prin inscrierea de date in stiva; daca programul ajunge cu executia - contorul de program - in zona respectiva, se citeste data salvata in stiva, ca un cod de program, se decodifica si, in functie de "*codul*" pe care il reprezinta - daca exista codul respectiv in setul de instructiuni - se "*executa*". Evident, programul este complet alterat. Problema este dificil de depanat, deoarece este foarte greu de detectat cauza unui asemenea "defect" de program (care, ca simptome, se poate manifesta similar si in cazul unor defecte ale hardware-ului - memorii defecte, de exemplu). Se poate ajunge la executarea "pas cu pas" a programului, si examinarea continutului memoriei program, pe parcursul executiei programului. O masura necesara este de a implementa o stiva cat mai mare cu putinta, avand o rezerva cat mai mare de memorie.

3.5.5 Limbajul de asamblare

Limbajul de asamblare utilizeaza, dupa cum s-a mentionat, *mnemonici* reprezentand prescurtari (*abrevieri*) ale denumirilor - semnificatiilor - instructiunilor unui anumit tip de microprocesor. Faptul

ca se asociază fiecărei instrucțiuni a microprocesorului câte o mnemonică, face ca programarea în *limbaj de asamblare* să reprezinte, după cum deasemenea s-a precizat, o programare în limbaj de nivel scăzut. Aceasta se datorează faptului că fiecare microprocesor are structură sa specifică (arhitectură, modalități de interfatare și moduri de adresare distincte), și corespunzător, seturile de instrucțiuni ale diferitelor tipuri de microprocesoare diferă între ele. Astfel, programatorul care scrie programe în limbaj de asamblare trebuie să stăpânească foarte bine atât elementele legate de *hardware*-ul cât și de *software*-ul microprocesorului respectiv. Pe de altă parte însă, programatorul ce utilizează limbajul de asamblare are la îndemână un instrument de lucru foarte puternic, în momentul în care îl stăpânește, deoarece doar prin utilizarea programării în limbaj de asamblare se poate avea un control absolut al procesorului. Principalul *dezavantaj* al utilizării limbajului de asamblare este că scrierea, depanarea și modificarea acestor programe este mult mai dificilă, și cere o foarte bună cunoaștere a microprocesorului și înțelegere a funcționării sale. O experiență bogată în acest sens este necesară pentru a implementa programe bune, eficiente.

Ușual, la ora actuală, la implementarea aplicațiilor industriale, utilizarea limbajului de asamblare se face concomitent cu utilizarea unui limbaj de nivel înalt. Astfel, *partile critice* ale programului (de obicei cele asociate operațiilor de timp real ale sistemului), necesitând o maximă eficientizare, mai ales din punctul de vedere al vitezei de execuție, se scriu în *limbaj de asamblare*. Restul programului, care conține părți ce se pot executa cu priorități mai scăzute, nu neapărat în timp real, poate fi implementat prin utilizarea unor limbaje de nivel înalt. Se poate astfel optimiza la maximum ca raport performanță/cost de producere, procesul de implementare a aplicației respective.

Pentru programul (partea de program) scris în limbaj de asamblare, faptul că este necesară implementarea elementară a programului, instrucțiune cu instrucțiune, face dificilă scrierea și testarea acestuia (prin defalcarea operațiilor sistemului în operații elementare, implementate cu câte o instrucțiune de program). Pe de altă parte, apare facilitatea de utilizare, gestionare a tuturor resurselor procesorului (registre, memorie, etc.), ceea ce poate conduce la optimizarea la maxim a programului.

Câteva reguli de bază la scrierea programelor în limbaj de asamblare sunt următoarele:

- programatorul răspunde de *completa gestionare a tuturor resurselor sistemului*; zonele de memorie ocupate de către program, cele alocate variabilelor, cât și cele alocate pentru stivă, trebuie definite explicit și respectate pe tot parcursul implementării programului. Similar, adresarea interfețelor de intrare/ieșire trebuie făcută corect, corelând pentru fiecare port în parte semnificația cuvintelor de date, cu acțiunile efectuate la adresarea acestora. O selecție greșită (adresare incorectă, sau a unei celule de memorie sau port inexistent), sau utilizarea incorectă a semnificației datelor, poate conduce la blocarea sau funcționarea defectuoasă a sistemului;
- în program trebuie specificate *adresele pentru fiecare variabilă, locație de memorie sau port accesate la un moment dat*;

- trebuie *anticipate si evitate* (prin modul de realizare a programului, sau prin implementarea de teste corespunzatoare) situatiile de aparitie a operatiilor cu rezultate imposibile (depasiri - *overflow* sau *underflow*);
- programatorul trebuie sa asigure *gestiunea valorilor (date)*, necesare la executia programului. Astfel, trebuie asigurata salvarea datelor (continutul registrelor de uz general, a celulelor de memorie cu utilizare multipla), in diferitele parti ale programului care utilizeaza aceleasi resurse ale sistemului;
- manipularea datelor la nivel intern se va face in cod binar, ceea ce va implica necesitatea unor *conversii de la/la nivelul utilizator* (valori zecimale, hexazecimale, caractere), *la nivelul masina* (binar);
- *depanarea programelor* se va face mai dificil, uzual utilizand metode speciale (de genul executiei pas cu pas a programului, si examinarii registrelor, continutului memoriei, etc.);
- in cazul unei *functionari defectuoase* a sistemului, este dificila localizarea defectului, care poate proveni din defecte pe partea *hardware*, sau din partea *software*, sau din combinatii ale celor doua componente, ceea ce necesita tehnici speciale, si o experienta bogata in testarea sistemului; uzual, prin programe simple se testeaza integritatea si completa functionare a *hardware*-ului, apoi se trece la implementarea unor programe complexe.

Elaborarea unui program in limbaj de asamblare nu este suficienta, dupa cum s-a mentionat, pentru obtinerea si implementarea formei finale a programului, in cod obiect, in memoria sistemului, si executia acestuia. Este necesar sa se dispuna de programul numit **asamblor**, care poate prelua *programul sursa*, scris in *limbaj de asamblare* (mnemonici), si poate genera codul obiect corespunzator fiecarei instructiuni. Scrise intr-o forma simbolica, asambloarele au o seama de facilitati suplimentare: comenzi, directive, care faciliteaza scrierea programelor; utilizarea de denumiri simbolice pentru variabilele programului; utilizarea etichetelor formale (in locul utilizarii adreselor efective ale variabilelor, subrutinelor, etc.); conversii automate din diverse baze de numarare in binar; posibilitatea definirii macroinstructiunilor ce permit simplificarea scrierii programului; etc. Totodata, programul asamblor verifica corectitudinea sintactica a programului scris in limbaj de asamblare, verifica corectitudinea adresarilor variabilelor din memorie, unicitatea lor, etc.

Programul asamblor poate fi rezident in memoria *PROM* a sistemului microprocesor, sau poate fi incarcat in memorie si apoi executat.

Pentru scrierea programelor sursa, continand programul in limbaj de asamblare, se utilizeaza deasemenea alte programe speciale, numite **editoare de text**, care permit scrierea efectiva a acestor programe, si memorarea acestora, intr-o zona a memoriei sistemului sau, la sistemele mai evolute, pe un suport de memorie externa de tipul discului flexibil, hard-discului, etc.

In fine, in etapa finala a elaborarii unui program, si mai ales in cazul necesitatii generarii programului executabil in forma sa finala, din mai multe parti de program, elaborate separat, (scrise eventual in

limbaje diferite, si compilate deasemenea individual, fiecare prin executia programului asamblor sau a compilatorului corespunzator), va fi executat un alt program specific, care realizeaza legarea acestor componente ale programului, numit **linker** (**link** = **legatura**). *Linker*-ul va face totodata alocarea efectiva a adreselor si resurselor sistemului, generand valorile corecte ale acestora in codul obiect final ce se va implementa si executa.

In etapa implementarii si testarii unui program, este deosebit de util un *program de depanare* (*debug*), prezent deasemenea la unele sisteme microprocesor.

Toate aceste programe necesare pentru a dezvolta, implementa si testa o aplicatie pe un sistem microprocesor, vor necesita ca, in etapa de creare a unei aplicatii, utilizatorul sa aiba la dispozitie un microcalculator de putere relativ ridicata, denumit **sistem de dezvoltare**. Pe aceste sisteme, programele se incarca in memoria *RAM* a sistemului, si pot fi cu usurinta testate, modificate, etc. De-abia dupa testarea completa, programele pot fi memorate definitiv in memorii de tip *PROM*, si implementate pe sistemele ce se vor utiliza in aplicatia finala, care eventual sunt dotate cu resurse mai reduse (mai putina memorie, periferice, etc.).

Din cele prezentate, reiese clar necesitatea utilizarii unei anumite strategii in elaborarea programelor pentru sistemele microprocesor.

3.5.6 Metodologia elaborarii programelor

Pasul important ce poate fi facut odata cu stapanirea setului de instructiuni al microprocesorului este elaborarea de programe, pentru a determina, prin lansarea lor in executie, implementarea aplicatiilor dorite pe sistemul microprocesor. Programatorul trebuie sa elaboreze un program corect din toate punctele de vedere, nici o eroare neputand fi acceptata. De aceea, in general, strategia elaborarii unui program de la punerea problemei, pana la utilizarea lui, va trebui sa treaca prin urmatoarele faze: **(1) analiza; (2) organizare; (3) codificare; (4) testare; (5) elaborarea documentatiei** aferente programului.

In **faza de analiza** se contureaza cu claritate cerintele programului, se elaboreaza expresiile matematice ce se vor utiliza. Un program bine elaborat se va ghida dupa un **algoritm**, care prevede si indica ce trebuie sa faca programul, in oricare dintre situatiile posibile ce apar in cursul rezolvarii problemei propuse. Astfel, *algoritmul* implica cel putin indicatii asupra actiunilor de urmat, intr-un numar finit de pasi, pentru o problema data. (Ca un exemplu clasic, etapele necesare pentru prepararea unei retete reprezinta un algoritm).

Strans legata de *faza de analiza* este **faza de organizare**. *Algoritmul* generat anterior este organizat in operatiile necesare pentru rezolvarea problemei. Este etapa in care se elaboreaza asa numitele **scheme logice** (diagrame ale algoritmului, utilizand niste simboluri standard, desemnand diversele actiuni posibile --- vezi fig.3.49).

De modul de elaborare optima a algoritmului si a schemei logice, ca organizare si eliminare a portiunilor redundante, depinde obtinerea unui program cat mai eficient (rapid si ocupand memorie cat mai mica). Nu este absolut necesara, pana in acest punct, cunoasterea setului de instructiuni. Totusi, pentru o utilizare cat mai eficace a posibilitatilor microprocesorului, cunoasterea elementelor de hardware si software specifice acestuia este de dorit.

Faza de codificare, care urmeaza, reprezinta momentul trecerii la elaborarea codului programului. Aici se utilizeaza setul de instructiuni. Pe baza lui, se scriu instructiunile ce vor genera actiuni ale microprocesorului, corespunzatoare algoritmului si schemei logice elaborate.

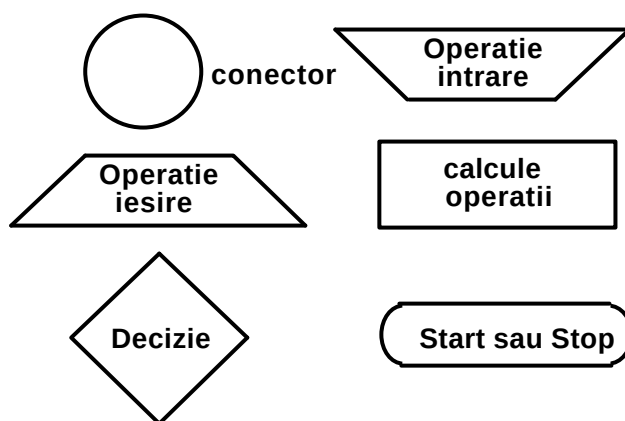


Fig.3.49. Simboluri tipice utilizate la alcatuirea schemelor logice

Dupa *codificare*, programul este mai intai **testat** prin comparare cu schema logica, verificand corectitudinea fiecărei instructiuni. Apoi, programul este introdus in memoria microcalculatorului (in cod masina), trecandu-se la **executia** lui. Este etapa in care se localizeaza si lamuresc diversele erori ce au putut aparea la una dintre fazele trecute.

Totodata trebuie remarcat faptul ca, adesea, aceste erori vor implica **revenirea** si **modificarea** unor parti din algoritm si/sau schema logica, si/sau codul elaborat. Programul va trebui executat de mai multe ori, pana la obtinerea variantei corecte si definitive.

In acest ultim punct, un pas deosebit de important este reprezentat de **faza de documentare**. Toate informatiile referitoare la etapele anterioare, de la punerea problemei, elaborarea algoritmului si a schemei logice (eventual diversele versiuni succesive, indicand evolutia programului), codul programului (cu comentarii suficiente pentru a se corela cu schema logica), eventuale esantioane de date de intrare/iesire, daca este cazul, trebuie sistematizate si incluse in documentatia aferenta programului respectiv.

Sensurile majore ale acestei faze sunt reprezentate atat de necesitatea de a putea rememora cu usurinta elaborarea programului (o succesiune de instructiuni necomentate devine, dupa cateva zile de la scriere, foarte greu de analizat, chiar de catre cel care a elaborat-o), cat si de posibilitatea utilizarii ulterioare a experientei castigate la scrierea programului respectiv, pentru eventuale modificari ale acestuia, sau la elaborarea unor programe noi.

Concluzionand , programarea microprocesoarelor reprezinta o problema serioasa, necesitand o buna pregatire si experienta, castigate prin scrierea de programe cu grad de dificultate crescut, corelata cu posibilitatea testarii acestora pe viu (pe un sistem microprocesor), ajutand la identificarea si eliminarea erorilor inerente ce apar. Ea se bazeaza pe cunoasterea, cel putin pana la un anumit nivel de profunzime, a arhitecturii de baza a sistemului, a setului de instructiuni al microprocesorului, a posibilitatilor si a metodelor de interfatare cu echipamente de intrare/iesire, cat si pe aplicarea unei anumite metodologii de programare. Aceste aspecte vor fi reluate in capitolele urmatoare, insotite de exemple corespunzatoare.

3.6 ELEMENTE DE INTRARE/IESIRE ALE SISTEMELOR MICROPROCESOR

Sistemul microprocesor trebuie sa interactioneze cu lumea inconjuratoare. Este necesara atat primirea de informatii, din afara acestuia, cat si, pe baza calculelor efectuate, furnizarea de informatii, comenzi etc., spre exterior.

Desi *unitatea centrala (CPU)* poate efectua multiple operatii interne, este necesara comunicatia acesteia cu exteriorul, fara de care functionalitatea sistemului ar fi practic inexistentă. Deoarece singura cale de comunicatie dintre procesor si exterior este reprezentata de *magistrala de date* a sistemului, (utilizand totodata magistralele de adrese si control), este necesar sa se utilizeze *circuite speciale* care sa *converteasca* informatiile diverse, de la o gama foarte larga de echipamente si componente de intrare si iesire, la semnale *compatibile* cu cele de pe magistralele sistemului microprocesor. Aceste *circuite*, care implementeaza o *functie de compatibilizare*, se numesc module de interfata (sau module de intrare-iesire). Astfel, comunicatia cu unitatea centrala reprezinta transferul datelor prin aceste interfete, realizata intr-un mod selectiv si controlat uzual (dar nu exclusiv - vezi tehnicile *DMA*), de catre microprocesor. *Conventiile* utilizate pentru a implementa aceasta comunicatie, incluzand temporizarile semnalelor, controlul procesului de transfer de date, modul de reprezentare (codificare) a datelor, alcatuiesc asa-numitul protocol al comunicatiei. Functia de baza a modulelor de interfata va fi aceea de a *converti* semnalele de pe magistralele CPU pentru seturile de *porturi de intrare sau iesire*. Un port reprezinta astfel o colectie de componente ale sistemului la care se pot conecta echipamente externe (*periferice*).

Dupa cum s-a precizat si in paragrafele precedente, gama arhitecturilor si a implementarilor diverselor sisteme microprocesor este extrem de larga, fara o standardizare a functiilor si/sau semnalelor magistralelor acestora. Pe de alta parte, aceste sisteme diverse vor trebui sa interactioneze cu gama, deasemenea larga si nu neaparat unitara, a echipamentelor periferice, utilizand o paleta importanta de semnale, viteze de lucru si protocoale de comunicatie. Compatibilitatea intre operatiile interne ale procesorului si aceste semnale externe se va asigura de catre o mare varietate de tipuri de *interfete*. Uzual, firmele producatoare de microprocesoare vor asigura si *circuitele specializate*, compatibile cu tipurile de microprocesoare produse de catre acestia, pentru a se implementa diversele *interfete specializate*.

Astfel, comunicatia între microprocesor si echipamentele externe are loc in *doua etape succesive*: prima, in care transferul are loc *între CPU si interfata*, si a doilea, in care comunicatia are loc *între interfata si periferic*. Uzual, *primul stadiu* se denumeste operatie de intrare-iesire, iar al *doilea*, operatie de transmisie a informatiilor (v.fig.3.50).

In consecinta, interfetele de intrare/iesire reprezinta calea de legatura ce asigura transferul de date între microprocesor si exteriorul lui.

Exista numeroase posibilitati pentru realizarea acestor legaturi. In functie de configuratia sistemului si de cerintele aplicatiei, se pot utiliza structuri mai simple sau deosebit de complexe.

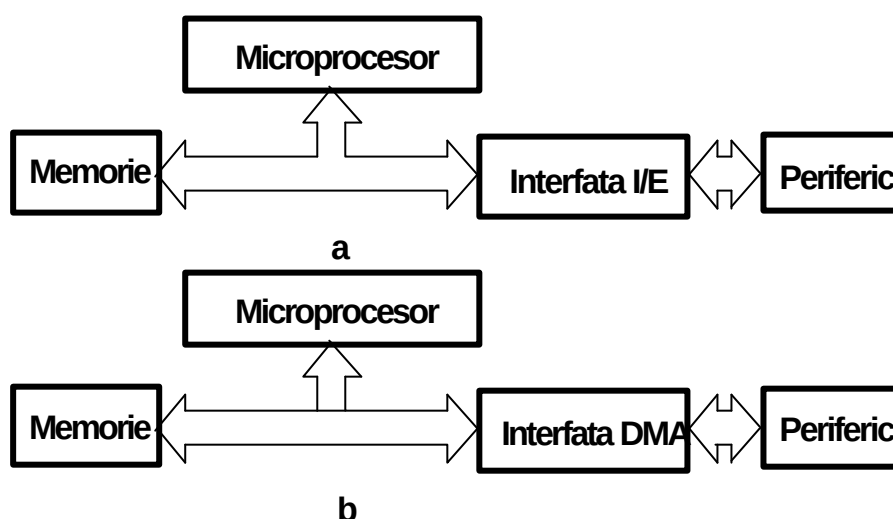


Fig.3.50. Moduri de interfatare: (a) sub controlul microprocesorului; (b) prin acces direct la memorie

Tinand seama ca elementele de intrare asigura preluarea informatiei dintr-o forma accesibila omului (reprezentari --- caractere, litere, cifre, pana la imagini tv.), sau de la elemente diverse ale instalatiilor industriale (semnalizari, traductoare etc.), respectiv cele de iesire furnizeaza informatii utilizabile de catre operatorul uman (mesaje tiparite, semnalizari audio-vizuale, sinteza de voce), cat si pentru comanda, actionarea elementelor echipamentelor industriale, se poate aprecia gama deosebit de larga de tipuri si configuratii in care se vor prezenta atat interfetele de intrare/iesire, cat si perifericele sistemelor microprocesor.

Semnale exterioare de o mare varietate a formelor de prezentare (analogice, numerice, de alta natura decat electrice), trebuie convertite la formele de informatie, limitate, acceptate de catre microprocesor. Conversii similare trebuie efectuate la iesirea spre exteriorul sistemului microprocesor. Trebuie adaptate deasemenea vitezele de lucru diferite ale perifericelor, respectiv ale microprocesorului. Uzual, acesta lucreaza mult mai rapid, putand efectua numeroase operatii, in timp ce, de exemplu, se preia un caracter de la o consola (prin apasarea unei clape a acesteia) si se pregateste intr-o forma adecvata citirii de catre microprocesor. De asemenea, exista si echipamente foarte rapide (discuri magnetice, de exemplu), care pot furniza/prelua informatii catre/de la memoria

sistemului, cu viteze mai mari decat cele ce pot fi obtinute daca transferul se efectueaza de catre microprocesor (sub comanda acestuia). In aceste cazuri, se utilizeaza asa-numitele tehnici de transfer al informatiei prin **acces direct la memorie (DMA¹)**, pentru care accesul intre interfata si memoria sistemului se face sub controlul unui circuit specializat, fara controlul programului microprocesorului. Se vor adopta solutii *hardware-software* care sa asigure o performanta optima a sistemului. In functie de cerintele acestuia, de numarul de elemente pe care le supervizeaza microprocesorul, se vor adopta diverse tehnici de lucru.

Exista trei moduri principale de lucru cu elementele de intrare/iesire, permitand controlul si sincronizarea transferului de date:

(a) Operatii de intrare/iesire efectuate sub controlul programului;

(b) Operatii de intrare/iesire prin intreruperi;

(c) Operatii de intrare/iesire prin acces direct la memorie.

Tipul de operatie I/E utilizata intr-o aplicatie va depinde de rata de transmitere a datelor, de intarzierea maxima cu care microprocesorul poate prelua/transmite datele din momentul disponibilitatii echipamentului care le vehiculeaza, de posibilitatea executarii, intercalat cu operatiile I/E, a altor operatii ale unitatii centrale.

Din punctul de vedere al programarii microprocesorului pentru *operatii de I/E*, acestea se pot efectua fie prin executarea de **instructiuni specifice de I/E**, fie prin **instructiuni proprii operarii cu elemente de memorie** (v.par.urmator). In orice caz, pentru diferentierea intre diversele echipamente ale sistemului, acestea sunt selectate in mod unic, conform informatiei de pe magistralele de adrese si de control, in mod asemanator selectiei celulelor de memorie.

Porturile sistemului vor prelua sau transmite informatia sub forma de *cuvinte de date*, de lungime specifica procesorului (8, 16 biti). Accesul la magistrala de date va fi facut tot prin *circuite tampon*. Astfel, tehnicile generale de *adresare si selectie de cip* ale *porturilor* vor fi similare celor de la *memorii*, cu unele particularitati specifice, care vor fi mentionate in paragrafele urmatoare.

Ca o *mentiune speciala*, trebuie remarcat ca, spre deosebire de memorii, unde la o adresa se va gasi o singura celula de memorie, aceeasi si pentru operatiile de scriere si pentru cele de citire, la operatiile cu *porturi*, este permis, si se si intalnesc situatii in care o *aceeasi adresa* este utilizata pentru *doua interfete diferite*, una de *intrare*, si alta de *iesire*. Aceasta deoarece pot exista porturi doar de intrare, si altele doar de iesire, deci cu flux unic al datelor, doar *de la* sau doar *spre* microprocesor.

In continuare se vor analiza cele trei tipuri de baza de organizare a transferurilor de date in operatiile de I/E ale microprocesorului.

3.6.1 Operatii I/E efectuate sub controlul programului

¹DMA: Direct Memory Access --- acces direct la memorie.

Una dintre cele mai accesibile si utilizate metode de efectuare a operatiilor de I/E este sub controlul nemijlocit al programului microprocesor. Astfel, prin instructiuni specifice, care determina selectia porturilor de I/E ale sistemului, microprocesorul efectueaza operatii de transfer de date intre aceste interfete si CPU.

Exista doua *tipuri de baza* (din punctul de vedere al semnificatiei), pentru informatia transmisa intre *microprocesor si portul de intrare/iesire*, si anume **cuvintele de control** si **cuvintele de date** (vezi fig. 3.51).

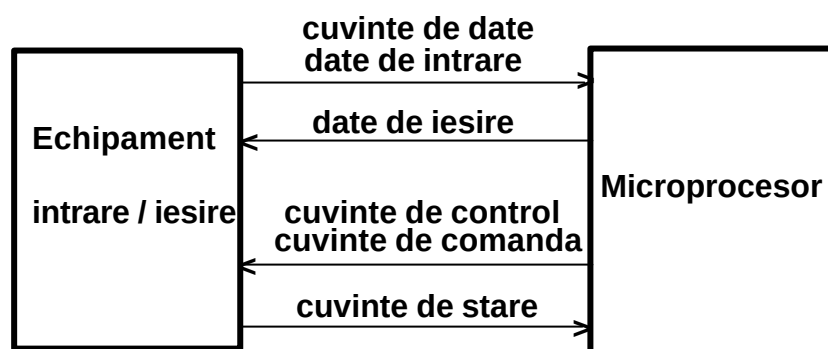


Fig.3.51. Fluxul de informatie intre microprocesor si o interfata de I/E

Cuvintele de control sunt utilizate pentru *sincronizarea* operatiilor portului si ale microprocesorului, si *setarea/testarea* unor parametri, operatii necesare in vederea transmiterii ***cuvintelor de date***.

Folosite in operatii de *scriere spre port*, *cuvintele de control* sunt denumite ***cuvinte de comanda*** a *portului* si permit, prin afectarea pentru fiecare bit al cuvântului, a unei semnificatii unice, *programarea portului* (pentru porturile care pot fi programate). Astfel se pot *initia* unele operatii sau *seta* unui parametri (oprirea unui motor, modificarea ratei de transmisie a unor date, stergerea unui indicator de eroare etc.).

Citirea cuvintelor de control, denumite in acest caz ***cuvinte de stare***, permite, prin testarea prin program a bitilor cuvântului citit de la port, *analizarea* starii acestuia (daca are sau nu informatie disponibila, daca poate prelua alt cuvânt de date, daca transferul s-a efectuat corect etc., in functie de posibilitatile si functiile portului) si luarea de *decizii* adecvate de catre microprocesor (in speta, se vor executa in mod corespunzator acele sectiuni din programul sistemului care trateaza situatia existenta).

La efectuarea *operatiilor de intrare/iesire sub controlul programului*, se pot utiliza:

(a) instructiuni specifice de I/E. In acest caz, microprocesorul este prevazut cu semnale pe magistrala de control, diferite pentru operatiile cu memoria, respectiv cu porturile. Se conecteaza atat memoria cat si porturile la magistralele de adrese si de date ale sistemului, dar se utilizeaza semnale de control distincte pentru cele doua categorii de componente ale sistemului. Figura 3.52 prezinta

modul de structurare a sistemului in acest caz. Utilizand instructiunile specifice de I/E, prin program, se genereaza la executia acestor instructiuni semnalele de control asociate, selectand porturile de I/E si efectuand transferul dorit al informatiilor.

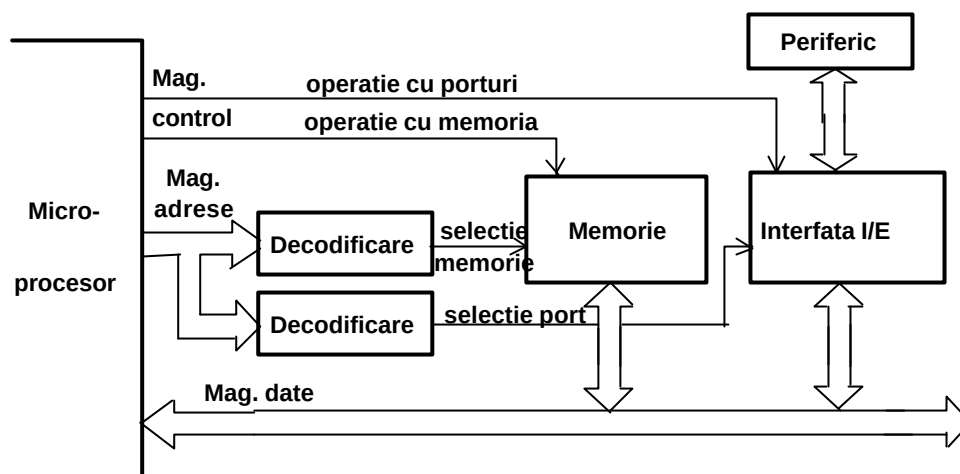


Fig. 3.52. Utilizarea operatiilor specifice de I/E, distincte de cele cu memoria

Exista doua moduri principale de implementare a acestui tip de operatii I/E:

(a1) cate o instructiune unica pentru fiecare operatie de I/E mentionata, **utilizand minimum o singura adresa** pentru un port. In acest caz, exista patru instructiuni tipice:

- (1) *citire de date (citire cuvant de date);*
- (2) *scriere de date (scriere cuvant de date);*
- (3) *transmitere comanda (scriere cuvant de comanda);*
- (4) *citire stare (citire cuvant de stare).*

Acest tip de operatie cu porturile ar necesita un numar de minimum 2 semnale de control, pentru a diferentia, la o operatie cu acelasi port, *sensul* si *tipul* instructiunii executate la un moment dat.

Figura 3.53 prezinta structura sistemului in acest caz. Metoda **nu** este utilizata in mod tipic, in sistemele microprocesor uzuale.

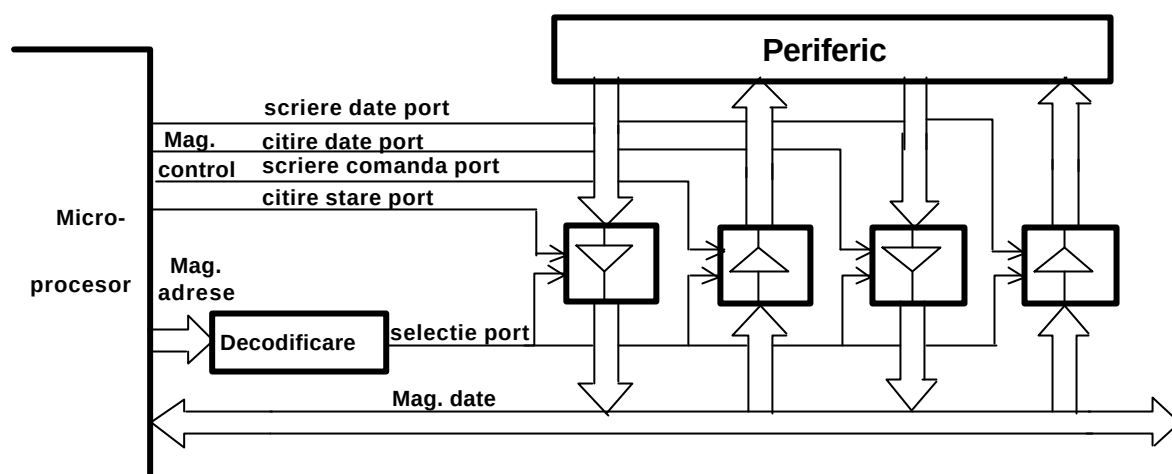


Fig. 3.53. Utilizarea unei singure adrese de port, si a patru semnale de control I/E

(a2) *doua instructiuni I/E*, una pentru *intrare*, alta pentru *iesire*, atat pentru cuvintele de date, cat si pentru cele de control. Se vor utiliza in acest caz **minimum doua adrese pentru un port**, pentru a *diferentia* operatia ce se efectueaza, referitor la *date* sau *control*. Cele doua instructiuni tipice, ce se definesc in acest caz, vor fi:

- (1) *citire informatie (cuvant de data sau de stare);*
- (2) *scriere informatie (cuvant de data sau de comanda).*

Pentru acest tip de operatie de I/E, este necesar minimum un semnal de control care sa diferentieze *sensul* transferului datelor (*citire* sau *scriere*), care corelat cu *adresa* de la / la care se face transferul, sa poata defini in mod unic operatia ce se executa. Metoda este una dintre cele mai intalnite tehnici de utilizare a porturilor de I/E. Figura 3.54 prezinta structura de principiu a sistemului in acest caz.

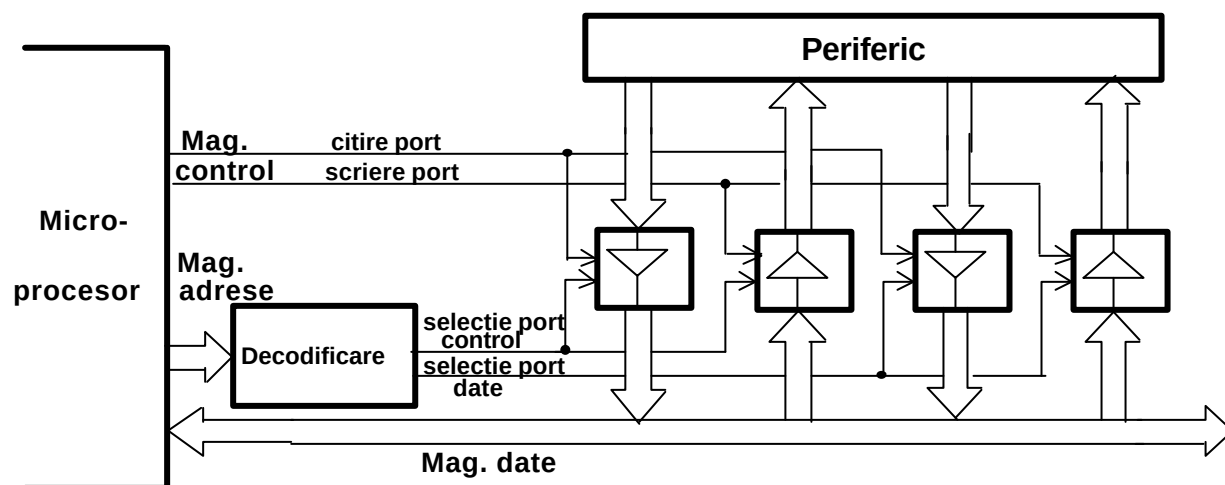


Fig. 3.54. Utilizarea a doua adrese I/E si a doua semnale de control

(b) nu se folosesc instructiuni separate de I/E. Se va **adresa portul ca o celula de memorie**. Se configureaza memorie in sistem, mai putina decat capacitatea maxima a acestuia. La adresele ramase libere se va asigura selectia portului respectiv. (Deci, la aparitia semnalelor de adresa corespunzatoare si a celor de control, proprii lucrului cu celula de memorie respectiva, se va selecta de fapt si se va vehicula informatie cu portul dorit)¹. Desi memoria sistemului devine astfel mai mica, acesta este un impediment minor, compensat in primul rand de faptul ca paleta de posibilitati oferita de instructiunile de tip lucru cu memoria este mult mai larga. (Din punctul lui de vedere, microprocesorul va efectua operatii cu o celula de memorie, nu cu un port). Operatiile tipice pentru a efectua transferul datelor vor fi in acest caz:

- (1) **incarcare data (citire cuvant de data sau stare);**
- (2) **memorare data (scriere cuvant de data sau comanda).**

Figura 3.55 prezinta structura de principiu a sistemului in acest caz.

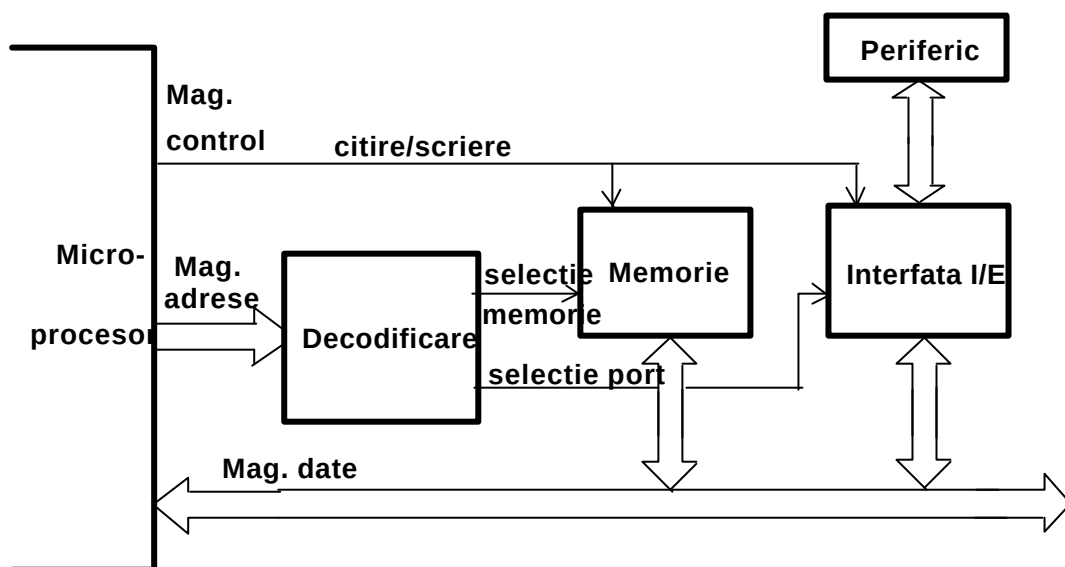


Fig. 3.55. Utilizarea operatiilor de I/E, de tipul "memory mapped I/O"

O comparatie succinta a celor doua metode prezentate anterior poate evidientia elementele esentiale ale acestora, de care se tine seama la configurarea structurii sistemului microprocesor. Astfel, utilizarea *instructiunilor si a semnalelor specifice de I/E*, are avantajul unei mai bune organizari si clarificari a programului, cat si a configurarii schemei de selectie a memoriei si porturilor sistemului. Pe de alta parte, metoda "memory mapped I/O" permite utilizarea instructiunilor uzuale de lucru cu memoria, cu avantajele aferente (mult mai multe instructiuni aferente, moduri de adresare complexe, etc.); pe de alta parte insa, metoda implica complicatii in ceea ce priveste schemele de adresare si selectie a memoriei si interfetelor sistemului. In plus, o problema suplimentara in acest caz o reprezinta modul si metodologia necesar a fi utilizata la *depanarea* programului, fiind mai dificila diferentierea operatiilor cu memoria de cele cu porturi, de vreme ce instructiunile utilizate la aceste operatii coincid (doar adresele diferentiind cele doua tipuri de componente).

¹ Metoda este desemnata cu termenul de "memory-mapped I/O" --- *intrare/iesire mapata in memorie*

Capacitatea de adresare a porturilor, pentru tipurile uzuale de microprocesoare, variaza de la 256 de porturi I/O (uzual la microprocesoarele de 8 biti), pana la 64k porturi de I/O sau mai mult. In cazul utilizarii porturilor organizate ca "memory mapped I/O", din totalul spatiului de memorie configurabil pentru microprocesorul respectiv se va decide combinatia *memorie/porturi* optima pentru aplicatia respectiva. Uzual, atat memoria cat si porturile sistemului se organizeaza in acest caz in *zone compacte de adrese (blocuri)* (pentru memorie acest lucru fiind, evident, absolut necesar).

Modalitatile de utilizare in program a *cuvantului de control* sunt, in principal, date de o secventa de program, de genul:

- (a) se scrie cuvantul de comanda la port, pentru a cere transferul de cuvant de date;
- (b) se citeste cuvantul de stare de la port;
- (c) se verifica bitii de stare ce indica posibilitatea transferarii datelor;
- (d) se reiau pasii b-c, pana ce portul este gata de transfer;
- (e) se citeste (scrie) cuvantul de date.

O astfel de secventa poate aparea pe parcursul programului principal al microprocesorului.

Cu aceasta mentiune, figura 3.56.a prezinta implementarea secventei de executare a operatiilor de I/E in cadrul programului principal, utila in realizarea programelor simple; in figura 3.56.b se prezinta implementarea acestei secvente sub forma unei subrutine, apelata in cursul programului principal, in punctele in care se doreste transferul de informatie de la/la portul respectiv, sau doar se testeaza starea acestuia. Se evita astfel pierderea din timpul de lucru al procesorului. Problema este si mai delicata in cazul mai multor porturi. Utilizarea a cate unei astfel de subrutine pentru testarea starii nu este avantajoasa; se implementeaza solutii de genul celor din figura 3.56.c. Microprocesorul poate testa rapid starea mai multor porturi, evitand testarea separata a acestora. Totusi, pot aparea intarzieri considerabile intre disponibilitatea portului si preluarea/transmiterea datelor de la/la acesta. Totodata, programul principal trebuie sa apeleze, in puncte stabilite de catre programator, si deci doar cand este disponibil (neavand alte sarcini mai importante), aceste subrutine.

O solutie mult mai avantajoasa, din punctul de vedere al timpului de raspuns la evenimente (cand este posibil transferul), care permite implementarea unor programe mai clar si corect elaborate, este oferita de catre efectuarea operatiilor I/E prin intreruperi.

3.6.2 Operatii de intrare/iesire efectuate prin intreruperi

Operarea prin **intreruperi** este bazata pe utilizarea unor semnale de control specifice, lansate de la periferice, catre microprocesor, anuntand un eveniment semnificativ la nivelul acestora (date disponibile pentru a fi preluate, posibilitatea acceptarii de noi date de la procesor etc.).

Lucrul in intreruperi permite *operarea aparent simultana a mai multor programe* distincte ale microprocesorului.

In principal, aceasta tehnica este aplicata pentru controlul si lucrul cu echipamentele periferice si pentru implementarea sistemelor ce opereaza ***in timp real*** (efectuand anumite operatii --- masuratori, calcule, comenzi), la intervale predeterminate de timp. Bazandu-se pe faptul ca, in general, procesorul are viteza de lucru net superioara perifericelor, el lucreaza cu fiecare doar la "*cererea*" acestuia, putand astfel sa serveasca mai multe dintre ele, la vitezele lor de lucru.

Sistemul de intreruperi va permite acest mod de solutionare a problemei. Intr-o forma simplificata, aparitia unei intreruperi la procesor determina urmatoarele evenimente:

- ***terminarea instructiunii curente a microprocesorului;***
- ***memorarea valorii contorului de program (PC) in stiva;***
- ***incarcarea PC cu o adresa de program, predefinita;***
- ***inhibarea intreruperilor si continuarea executiei de la noua adresa a PC.***

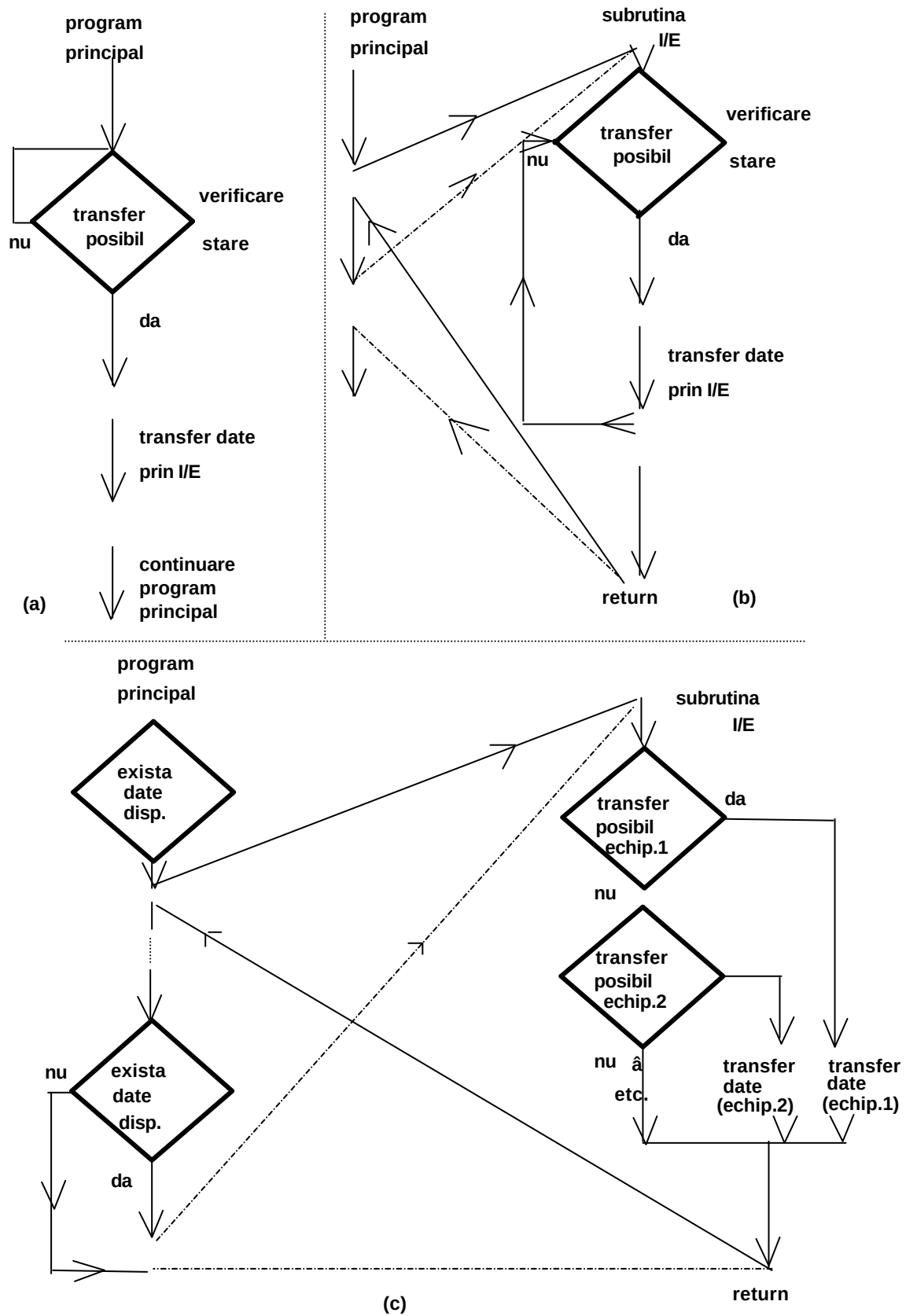


Fig.3.56 Operatii I/E controlate prin program

Dupa tratarea intreruperii (zona de program la care s-a sarit la intrerupere este scrisa special pentru evenimentul aparut), se executa o instructiune specifica de revenire in programul intrerupt (se reface PC din stiva), continuandu-se executia acestuia (printr-un proces asemanator celui intalnit la revenirea din subrutinele obisnuite, apelate prin program). Figura 3.57 prezinta un exemplu in acest sens.

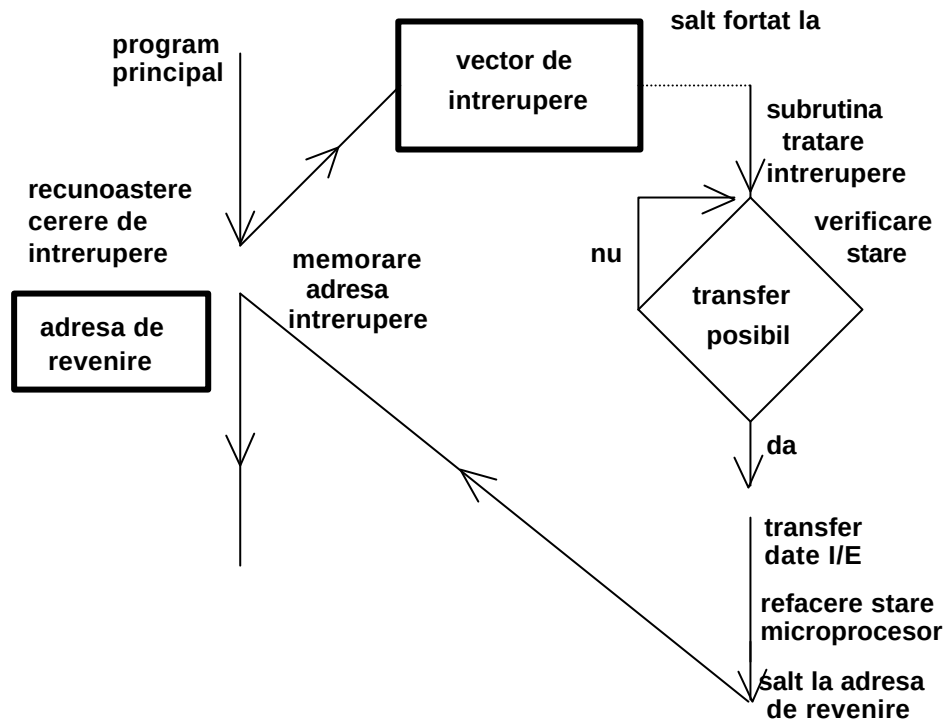


Fig. 3.57. Operatii I/E controlate prin intreruperi

Inhibarea intreruperilor, la acceptarea cererii de intrerupere, evita tratarea unei noi intreruperi pana ce nu s-a rezolvat cea in curs de tratare. Uzual, sistemul permite existenta *mai multor intreruperi*, de la diverse echipamente. Exista diverse *modalitati* pentru a determina saltul la adresa subrutinei aferente intreruperii respective. Astfel, este posibila existenta *mai multor pini distincti ai procesorului*, fiecare corespunzator unei anumite intreruperi. Aparitia pe unul dintre acestia a unui semnal activ va implica *saltul programului curent, la o adresa fixa, cunoscuta de catre programator*, si unde a fost plasata subrutina de tratare a intreruperii respective.

In cazul existentei unei *singure linii de intrerupere* la microprocesor, exista *doua moduri* principale de recunoastere a sursei care a generat intreruperea:

(a) Scanarea echipamentelor (polling). Saltul in subrutina de tratare a intreruperii este la adresa fixa. Aici se testeaza *cuvantul de stare* al fiecarui echipament de I/E, pentru a se determina, prin program, care echipament de I/E a generat intreruperea. Figura 3.58 prezinta aceasta tehnica, ce permite deci recunoasterea software a sursei de intrerupere.

(b) Vectori de intrerupere. In aceste structuri, in momentul in care microprocesorul este disponibil pentru acceptarea intreruperii, el confirma acest accept, setand un semnal catre echipamentele ce solicita intreruperea. Primind acceptarea, acestea vor genera pe magistrala de date, un *cod de*

instrucțiune de apelare de subrutină, cod generat de către *logica de întrerupere* (uzual, un **controler de întrerupere** --- circuit specializat pentru arbitrarea și generarea întreruperilor). Acest cod conține câte o adresă pentru fiecare echipament în parte. În general, aceste adrese sunt localizate secvențial în memoria program și formează așa-numitul **vector de întrerupere** al sistemului. (De exemplu, vor fi plasate în memorie, din 8 în 8 celule de memorie). Evident, cele 8 celule nu permit, de regulă, tratarea corespunzătoare a întreruperii. De aceea, programatorul va plasa aici doar o *instrucțiune de salt*, către adresa unde se găsește **subrutină** corespunzătoare **de tratare a întreruperii**).

O problemă importantă, la tratarea întreruperilor, este reprezentată de **prioritatea** cu care se execută acestea. Este posibilă apariția *simultană* a mai multor întreruperi, sau apariția unei întreruperi ce trebuie deservită, chiar în timp ce se deservea o întrerupere mai puțin urgentă. *Inhibarea*, pur și simplu, a întreruperilor elimină această posibilitate, ceea ce poate să nu ofere performanțe satisfăcătoare sistemului. De aceea, acordarea unei priorități fiecărei cereri de întrerupere și soluționarea acestora conform priorității avute sunt foarte importante.

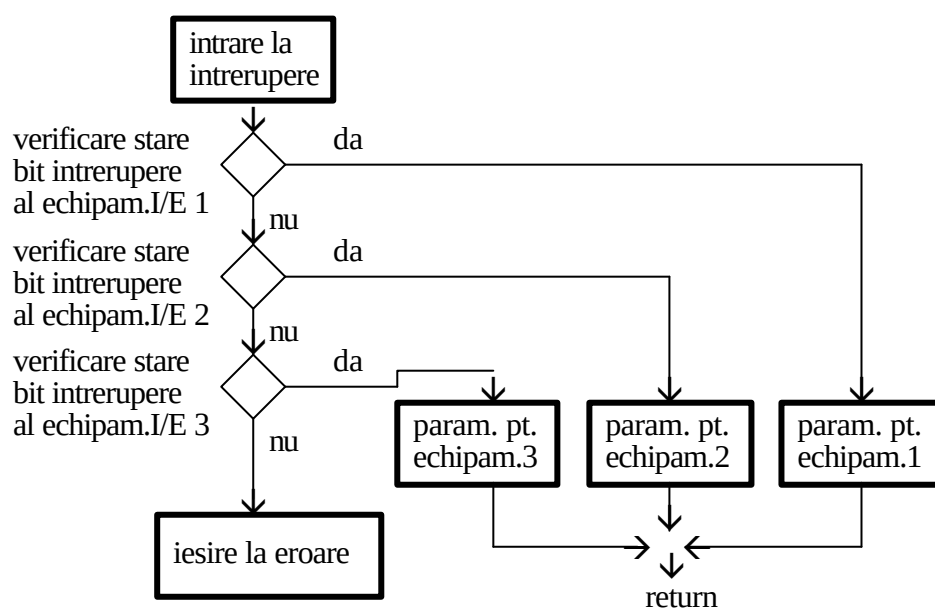


Fig. 3.58. Testarea cererii de întrerupere a echipamentelor de I/E în rutina de întrerupere

Se poate rezolva problema priorității atât prin *software*, cât și prin *hardware*.

(a) Priorități software. Ordinea în care se testează starea echipamentelor de I/E (conform fig. 3.58) corespunde priorității acestora. Detectarea, în ordinea testării, a unei cereri de întrerupere implică saltul la rutina corespunzătoare acestei întreruperi, care va avea prioritate.

(b) Priorități hardware. Un semnal de control trimis de către microprocesor în exterior trece prin logica de control a fiecărui echipament (vezi fig. 3.59). Dacă întreruperile sunt *inhibate (mascate)*, acest semnal nu permite generarea de întreruperi către procesor. Când întreruperile sunt *activate*, semnalul trece pe rând prin echipamentele ce nu cer întrerupere. Ajungând la cel care cere întrerupere, logica implementată blochează generarea de întreruperi de la următoarele echipamente și

determina generarea intreruperii catre procesor. Astfel, pozitia in lantul de echipamente determina prioritatea acestora.

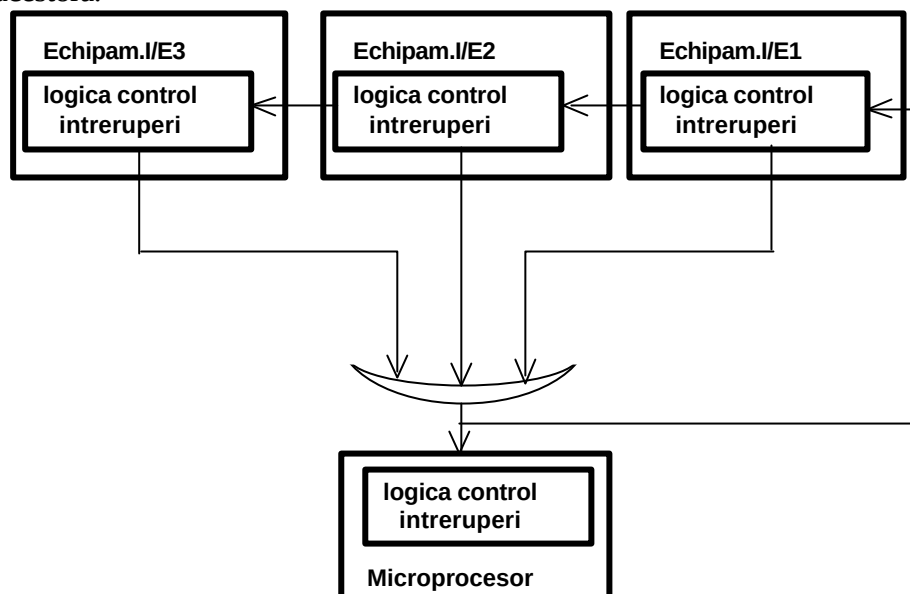


Fig.3.59. Implementarea hardware a logicii de prioritati la intrerupere, de la echipamentele I/E

Schemele cele mai sofisticate permit existenta a cate unui bit de mascare a intreruperii corespunzator fiecarui nivel de intrerupere existent. Conform acestor scheme, aparitia unei intreruperi determina mascarea nivelelor de prioritate inferioara. Intreruperi de prioritate mai mare vor intrerupe, la randul lor, subrutinele de tratare a intreruperilor de nivel inferior. Dupa tratarea evenimentelor prioritare, se va reveni la tratarea celor mai putin prioritare. Intreruperile aparute pe nivele inferioare celui tratat vor astepta (fiind mascate), terminarea tratarii curente. Figura 3.60 prezinta modul de desfasurare a evenimentelor in acest caz.

O mentiune trebuie facuta si referitor la **salvarea starii programului**, la aparitia unei intreruperi. Saltul la rutina de intrerupere va implica alterarea, prin operatiile efectuate acolo, a informatiei din registrele procesorului.



Unele microprocesoare au *doua seturi de registre*, unul pentru programul principal, altul pentru rutinele de intreruperi. Alte procesoare au *zona de registre de lucru in memoria RAM* a sistemului, exterioara microprocesorului. *Adresa de baza* a zonei de registre este continuta intr-un registru special al procesorului. Modificarea continutului acestuia implica operarea in alta zona de memorie. Refacerea valorii sale implica revenirea la vechea stare a programului (evident, trebuie luate masuri ca celelalte rutine sa nu afecteze, in cursul efectuarii diverselor operatii, aceste zone de memorie).

3.6.3 Operatii de I/E efectuate prin acces *direct la memorie*

3 - 89

Operatiile DMA se intrepatrund cu operatiile normale ale sistemului. Exista trei metode uzuale pentru transferuri DMA. Una dintre ele opreste microprocesorul, pe cand celelalte doua interfereaza cu operatiile sale curente.

Metoda de oprire a activitatii procesorului (HALT) implica, la aparitia unei cereri de acces DMA, terminarea efectuarii instructiunii curente a procesorului, urmata de trecerea tuturor pinilor acestuia (pe magistralele de date, adrese si control) in starea de mare impedanta si permite efectuarea transferurilor DMA pe aceste magistrale. Dezavantajul metodei consta in faptul ca vor fi necesare cateva cicluri-masina din momentul aparitiei cererii de transfer, pana la posibilitatea satisfacerii acesteia. Uneori, aceasta intarziere poate fi prea mare. Avantajos este faptul ca, odata obtinut, controlul magistralelor poate fi mentinut pentru transferuri oricat de lungi. Figura 3.61 prezinta aceasta tehnica de transfer DMA.

O alta metoda consta in **oprirea microprocesorului in cadrul instructiunii curente, la terminarea ciclului de instructiune curent**. Se obtin timpi de raspuns mult mai buni, deci performante sporite. Se pot transfera deasemenea seturi mari de date, la o astfel de intrerupere a executiei programului.

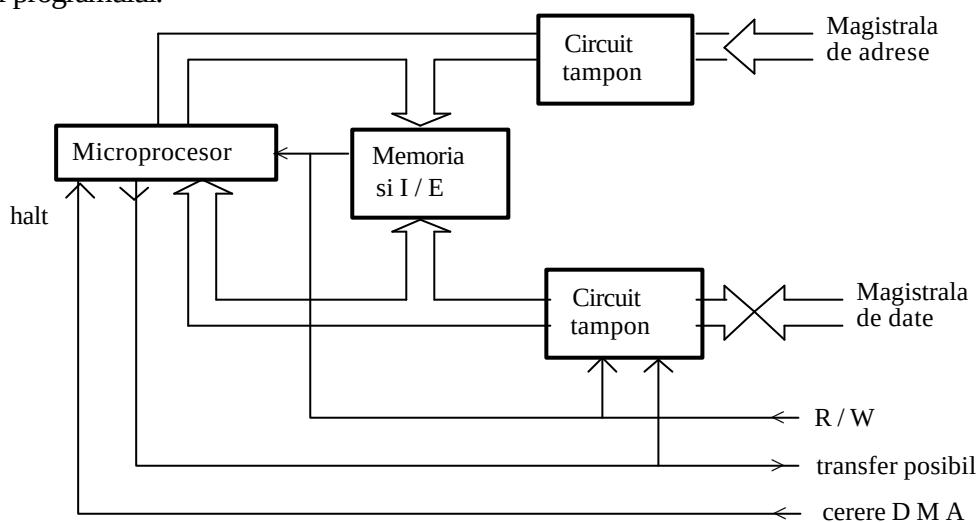


Fig.3.61. Transfer DMA cu oprirea microprocesorului la terminarea executarii instructiunii curente

Figura 3.62 prezinta schema tipica a sistemului in acest caz. Metoda lungeste practic durata executarii unor instructiuni (in timpul carora procesorul este oprit si se efectueaza transferul de informatii prin DMA).

A treia metoda este complet "transparenta" pentru procesor. Acesta nu este oprit, si nici incetinit la executarea programului. Este metoda cea mai rapida, dar si cea mai critica in acelasi timp. Necesita o deosebita atentie la proiectare si memorii cu timpi de acces foarte mici. **Operatiile microprocesorului si ale DMA sunt multiplexate**, luand in considerare faptul ca transferurile unitatii centrale apar doar pentru anumite faze ale ceasului sistemului.

Utilizarea unei anumite tehnici *DMA* se va face printr-un compromis între complexitatea structurii *hardware*, viteza de execuție necesară și rata de transfer *DMA* ce trebuie obținută.

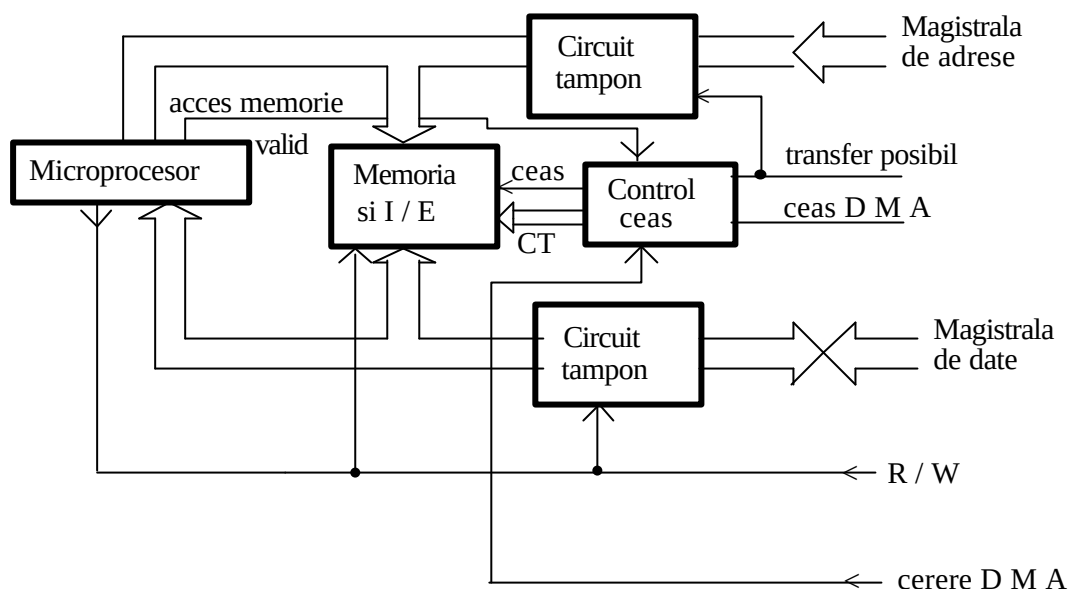


Fig.3.62. Transfer DMA cu oprirea microprocesorului la terminarea subciclului curent de instrucțiune

3.7 SISTEM MICROPROCESOR --- EXEMPLU

Pe baza conceptelor referitoare la *hardware*-ul și *software*-ul sistemelor microprocesor, prezentate în paragrafele anterioare, se va implementa în continuare un sistem microprocesor, într-un exemplu complet, atât ca structură *hardware*, cât și ca program, pentru o aplicație dată.

În ideea de a nu complica prezentarea cu detalii care ar dauna coerenței paragrafului, scopul urmărit fiind de a se concretiza modul de implementare al acestor sisteme, se va considera un microprocesor ipotetic, nu dintre tipurile existente, ci cu unele simplificări, la prezentarea structurii *hardware* a sistemului. Menționăm însă că partea de *software* este implementată în limbajul de asamblare al unui microprocesor de 16 biți (compatibil 8086).

Aplicația aleasă, fiind doar un exemplu simplu, nu va acoperi nici pe departe posibilitățile reale ale unui microprocesor, atât din punct de vedere al *hardware*-ului, cât și din cel al *software*-ului. Dar nu acesta este scopul acestei prezentări.

Deci, care este **problema**: *sistemul microprocesor să realizeze liniarizarea ieșirii unui element neliniar (un traductor, de exemplu), generând, pe baza achiziționării marimii analogice de la ieșirea traductorului, un semnal analogic corespunzător, liniarizat*. Altfel spus, sistemul va trebui să *genereze un semnal analogic (ieșirea)*, având o valoare depinzând de un *semnal analogic de intrare*, dependentă, neliniară, fiind cunoscută și memorată în memoria sistemului, sub forma unui *tabel de valori*.

Din considerente de simplitate, s-au utilizat, atat pentru conversia *analog-numerica*, cat si pentru cea *numeric-analogica*, convertoare de 8 biti. Deoarece exista 256 de valori posibile ale marimii de intrare, s-au considerat existente si memorate intr-un tabel de 256 de elemente valorile corespunzatoare pe care poate sa le ia marimea de iesire.

Pentru a sugera posibilitatea ca aceasta functie a sistemului procesor sa reprezinte doar una din sarcinile sistemului, s-a utilizat lucrul in intreruperi, iar portiunea de program implementand functia dorita a fost considerata o subrutina de intreruperi. In acest context, doar la momente de timp corespunzand disponibilitatii datelor de intrare (de la *CA/N*, la sfarsitul conversiei analog-numerice), procesorul va fi anuntat de catre acesta, prin *intrerupere*, va executa citirea, prelucrarea datei si va da catre *CN/A* valoarea corespunzatoare intrarii. Terminarea intreruperii va permite revenirea la executarea altor sarcini ale sistemului.

3.7.1 Structura hardware a sistemului

Schema de principiu care va implementa aplicatia dorita este prezentata in fig.3.63. Trebuie sa mentionam ca, pe baza aceleiasi scheme, modificarea programului va permite implementarea unor functii noi (sistemul poate functiona, de exemplu, si ca regulator etc.).

S-a presupus, dupa cum se observa, ca s-a utilizat un microprocesor de 8 biti, cu o magistrala de adrese de 16 biti. Se considera disponibile direct la iesirea microprocesorului semnalele: $\overline{IO/M}$ (indicand operatie cu un port, daca este 1 logic, sau cu memoria, daca este 0 logic); $\overline{WR}$ (operatie de scriere, pentru 0 logic); $\overline{RD}$ (operatie de citire pentru 0 logic); $\overline{INTA}$ (acceptarea unei cereri de intrerupere, pentru 0 logic). La intrari, se considera semnalele: $\overline{RESET}$ (care pentru 0 va aduce procesorul in starea de start initial; se considera in acest caz, ca PC va fi initializat cu valoarea 0); $\overline{READY}$ (indicand procesorului ca transferul de date de la/la memorie sau port s-a efectuat); $\overline{INT}$ (semnal care, la valoarea 1 logic, va indica cerere de intrerupere catre procesor). Tot din considerente de simplitate, s-a presupus existenta unui singur nivel de intrerupere, la adresa, fie, 0004H, fixa, valoare pe care contorul de program (*PC*), o va primi in mod automat, la acceptarea unei cereri de intrerupere.

Ca o *mentiune cu caracter general*, legata de simbolistica utilizata in prezentarea schemelor, reamintim ca *semnalele barate* sunt *active* pentru valoarea 0 logic, cele *nebarate* sunt *active* pentru 1 logic. Totodata, simbolul de inversare este utilizat in scheme, in conformitate cu functia logica implementata de catre circuit, din punct de vedere al logicii semnalelor de intrare si al celor de iesire din circuit. Astfel, la inversarea semnalului $\overline{IO/M}$, pentru a genera semnalul $M/\overline{IO}$ folosit la selectarea memoriilor, se remarca utilizarea simbolului de inversare la intrarea si nu, cum era uzual, la iesirea inversorului. Astfel se accentueaza faptul ca un semnal 0 pe intrare va fi 1 pe iesire, aceasta fiind situatia necesara la selectie de memorie. Aceeasi remarca si la sinteza selectiei de cip pentru circuitele *PROM*: semnalul $\overline{CSP0}$, de exemplu, va aparea la coincidenta in valoare logica 0 a semnalelor $\overline{RD}$ si $\overline{BANC0}$ (un SI-NU de semnale active pe 0), deci dat de functia logica:

$$\overline{\text{CSP0}} = \overline{\text{RD}} \cdot \overline{\text{BANC0}} = \text{RD} + \text{BANC0}$$

implementata fizic cu un circuit de tip *SAU*. Utilizarea simbolului pentru circuitul *SAU* nu ar fi fost suficient de clara pentru a intelege direct functionarea logica a schemei.

Pentru memoria program s-au utilizat doua circuite *PROM* de cate 256x8 biti capacitate, suficient, dupa cum se va vedea, pentru aplicatia data. Memoria *RAM*, de 256 cuvinte, s-a obtinut prin utilizarea a 2 cipuri de 256x4 biti fiecare, functionand in paralel, unul pe primii 4 biti de date (D0÷D3), celalalt pe ultimii 4 (D4÷D7). Profitand de utilizarea memoriilor de 256 octeti, s-a implementat logica de selectie a memoriei din bitii superiori de adresa (Ad8 ÷ Ad15), obtinandu-se semnalele $\overline{\text{BANC0}}$, $\overline{\text{BANC1}}$, $\overline{\text{BANC2}}$, corespunzator zonelor de adrese 0÷0FFH, 100H÷1FFH, respectiv 200H÷2FFH. Astfel, memoria program se va gasi incepand de la adresa 0 (de la aceasta adresa se lanseaza procesorul, la RESET), pana la adresa 1FFH, iar memoria *RAM*, intre adresele 200H si 2FFH.

Se remarca selectia memoriei *PROM* doar la operatii de citire (corelat cu semnalul $\overline{\text{RD}}$). Bitii inferiori de adresa Ad0÷Ad7 sunt utilizati direct la nivelul cipurilor, pentru selectia celulelor de memorie de pe cipul selectat.

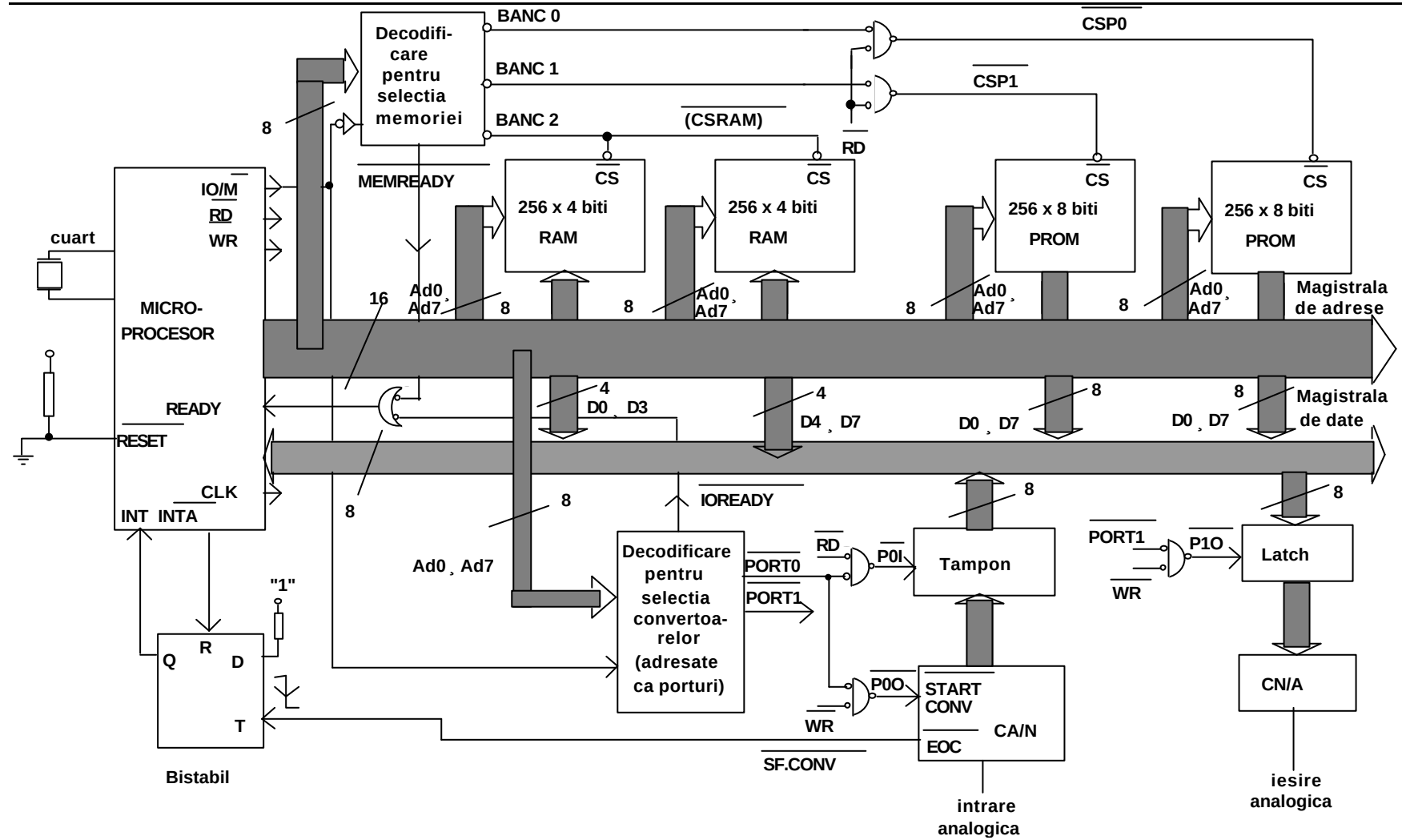


Fig. 3.63. Schema de principiu a sistemului microprocesor exemplificat

În ceea ce privește adresarea convertoarelor, s-a considerat că sunt elemente de tip porturi, adresate prin instrucțiuni de I/E. Se consideră că microprocesorul poate adresa un număr de 256 porturi de intrare, și 256 porturi de ieșire, deci la o operație de I/O, denotată de apariția semnalului $\overline{IO/\overline{M}}$ de valoare 1, primii 8 biți de pe magistrala de adrese conțin adresa portului cu care se va opera. Corelat cu semnalele de $\overline{RD}$ sau $\overline{WR}$, se poate cunoaște și sensul transferului de informație (input sau output). (Nu este necesară nici o legătură funcțională între un port de intrare și unul de ieșire, care au aceeași adresă, diferența constând în selectarea unuia printr-o operație de *INPUT* --- semnal $\overline{RD}$, respectivă celuilalt printr-o operație de *OUTPUT* --- semnal $\overline{WR}$).

Convertorul *A/N* considerat are un pin de comandă $\overline{START.CONV}$, pe care, aplicând un semnal 0 logic, se determină startul unei conversii analog-numerice a mărimii de la intrarea convertorului. Durata conversiei este de ordinul 100 microsecunde --- nu considerăm un convertor prea rapid. Tot din considerente de simplitate, se presupune că semnalul de intrare variază lent comparativ cu durata conversiei (pe timpul căreia se consideră constant), altfel fiind necesară utilizarea unui circuit de esantionare și reținere (*SAMPLE AND HOLD*). La sfârșitul conversiei, convertorul va genera semnalul $\overline{EOC}$ (*End of Conversion - sfârșit de conversie*).

În configurația aleasă, semnalul va determina setarea unui bistabil, generând un semnal de întrerupere către procesor. Printr-o operație de citire de la portul P0 (fie la adresa 00H), microprocesorul va avea acces, prin deschiderea circuitului tampon, la valoarea convertită de la ieșirea *CA/N*. Se consideră că startul conversiei se va da printr-o operație de *OUTPUT*, la portul P0.

În ceea ce privește conversia numeric-analogică, se va selecta la o operație de *OUTPUT*, la adresa 1H (portul P1), un circuit latch, care va memora informația corespunzătoare de pe magistrala de date. Ieșirile acestui circuit vor fi aplicate la intrările *CN/A*. Conversia *N/A* durează de ordinul sutelor de nanosecunde.

Se remarcă în schema sintetizarea semnalului de *READY*, din două semnale furnizate de către logica de selecție a memoriei, respectiv a porturilor. Se consideră că timpul de selecție al memoriei este inferior celui critic pentru a opera fără perioade de *WAIT*. Pentru operațiile cu porturile ($\overline{START.CONV}$, citire valoare marime convertită *A/N*, output o nouă valoare la *CN/A*), se consideră de asemenea că nu se pune problema timpilor de *WAIT*.

Semnalul $\overline{INTA}$, generat de microprocesor, în cazul acceptării unei întreruperi, va determina resetarea bistabilului care a generat întreruperea și anularea cererii de întrerupere, până la semnalizarea unui nou sfârșit de conversie.

3.7.2 Structura software a sistemului

Structura programului care va implementa aplicatia dorita este deosebit de simpla. Programul principal va trebui doar sa genereze semnalul de start pentru prima conversie, sa initializeze si sa activeze intreruperile. Apoi, microprocesorul va intra in starea de *HALT*, din care poate fi scos doar la intrerupere (sau la *RESET*, dar nu acest caz intereseaza). La aparitia unei intreruperi, anuntand sfarsitul conversiei *A/N*, se va trece la tratarea intreruperii. Subrutina scrisa in acest scop va salva starea microprocesorului (continutul registrelor). (Subrutina este scrisa pentru cazul general cand procesorul ar fi intrerupt din executia unui program, in care, revenind la sfarsitul intreruperii, trebuie sa poata continua executia normala). Apoi se va citi valoarea convertita. Considerand ca elementele tabloului cu valori ale functiei de iesire sunt aranjate la adrese succesive in memoria *PROM*, incat elementul corespunzator valorii *n* citite la intrare (*n* ia valori intre 0 si 255 --- marime pe 8 biti), va fi al *n*-lea in tabloul memorat, adresa acestuia va fi data de suma intre adresa primului element al tabloului si valoarea *n* citita. Odata obtinuta aceasta adresa, se va extrage din tabel valoarea corespunzatoare si se va transmite convertorului *N/A* prin operatia de *OUTPUT* respectiva. Se va da apoi un nou start de conversie la *CA/N* pentru o noua achizitie. Dupa refacerea starii microprocesorului dinainte de intrerupere (refacerea continutului registrelor) si reactivarea intreruperilor (acestea se dezactiveaza automat la acceptarea intreruperii), se revine in programul intrerupt (se iese din stiva, prin executia unei instructiuni de *RETURN*, valoarea *PC* salvata la acceptarea intreruperii --- in acest caz, adresa primei instructiuni dupa cea de *HALT*). Schema logica a subrutinei de intrerupere este prezentata in figura 3.64.

Exemplul fiind fictiv, dupa cum s-a mentionat deja, utilizam doar pentru convenienta, instructiuni specifice microprocesorului 8086 (ca mnemonica si semnificatie de principiu), fara a implementa in acest caz, un program riguros corect dedicat acestui microprocesor.

In continuare vom face cateva precizari inainte de a prezenta programul scris in limbajul de asamblare al unui asemenea microprocesor - de tipul *INTEL 8086* (sau, echivalent, microprocesorul *8088*, care are magistrala de date exterioara cipului, de 8 biti, iar interior opereaza pe 16 biti). Pentru intelegerea instructiunilor utilizate (comentate de altfel), reamintim ca microprocesorul *8086* (*8088*) are un registru acumulator *AX*, de 16 biti (care contine un operand, si rezultatul, la operatii aritmetice sau logice; la operatii de *INPUT* contine data citita de la portul adresat; la operatii de *OUTPUT* continutul sau este scris la portul adresat pentru iesire).

Exista deasemenea si registre utilizabile ca registre de uz general, cum ar fi *BX*, *CX*, de cate 16 biti, etc. (Pentru detalii asupra microprocesorului 8086, v. cap. 4). Toate aceste registre pot fi adresate global, ca registre de 16 biti, dar si individual, ca registre de 8 biti (*AL* si *AH*, ce compun registrul *AX*; *BL* si *BH*, ce compun registrul *BX*; *CL* si *CH*, ce compun registrul *CX*, etc.). Instructiuni specifice pot fi utilizate pentru a adresa si manipula continutul acestor registre, in operatii pe 8 biti. Deoarece exemplul dat este fictiv, respectam in cele ce urmeaza mnemonicele si sensul primar al instructiunilor similare ale microprocesorului 8086. Detaliile si elementele specifice acestui microprocesor se vor prezenta doar in capitolul urmator (segmentarea memoriei, moduri de adresare, tratarea intreruperilor, etc).

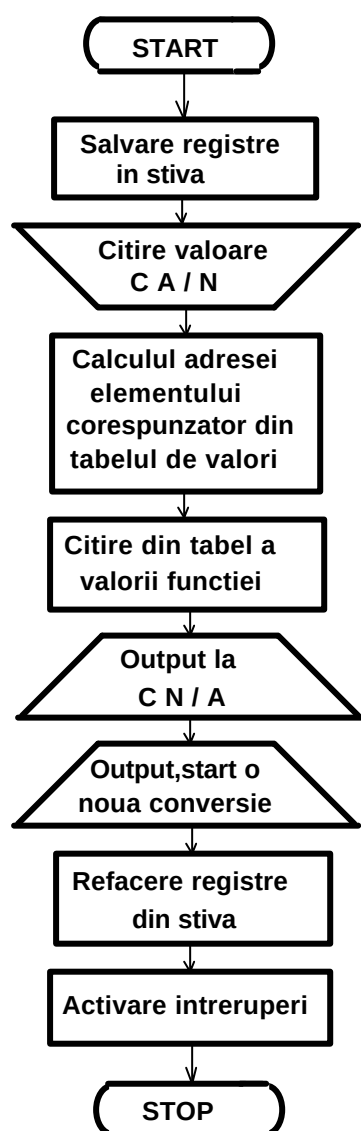


Fig.3.60. Schema logica a subrutinei de tratare a intreruperii la sfarsitul conversiei analog-numerice

Mai remarcam faptul ca trebuie avuta in vedere valoarea adreselor diferitelor parti din program (*zona de cod*, continand programul propriu-zis, si tabelul cu valorile functiei, situate uzual in memoria *PROM*, precum si *zona de date*, necesara in acest caz doar pentru crearea stivei programului, situata in memoria *RAM*).

La *RESET*, presupunem ca microprocesorul isi initializeaza *PC* cu 0. Deasemenea, se presupune, in mod simplificat, si fara a preciza mecanismele complete ale procesului de *intrerupere*, saltul la adresa 0004H, la aparitia unei cereri de intrerupere. Deoarece programul principal incepe la adresa 0H, la aceste doua adrese se vor gasi instructiuni de salt neconditionat, la programul principal propriu-zis si, respectiv, la subrutina de tratare a intreruperii.

In afara instructiunilor in limbaj de asamblare, carora le corespunde cod masina pentru microprocesor, se utilizeaza si asa-numitele *directive* pentru asambilor. Acestea nu determina generare de cod. Ele dau indicatii asambilorului, despre adresa la care sa se amplaseze o zona de program, valoarea unor simboluri literale in program etc. Deasemenea, asambilorul permite utilizarea adreselor simbolice (asemanatoare etichetelor instructiunilor in limbajele de nivel inalt), ceea ce usureaza efortul programatorului, eliminand si erorile posibile; de exemplu, o secventa de salt la o adresa se scrie:

...

instructiuni

JMP ADR1
instructiuni

...

ADR1: instructiune

...

Asamblorul va calcula, stiind lungimea (in cuvinte de instructiune, octeti), a fiecarei instructiuni, valoarea reala a adresei **ADR1**, si va genera codul corect de salt la aceasta adresa.

Observatie Textele scrise dupa semnul ";" sunt comentarii, neluate in seama de catre asamblor. Ele ajuta la intelegerea cat mai rapida a programului.

Listingul programului pentru implementarea aplicatiei propuse:

```

=====
;
;   Program test pentru citirea unui semnal analogic (utilizand un convertor
;   A/D), gasirea unei valori corespunzatoare unei dependente functionale,
;   prin cautare intr-un tabel de valori memorate, si transmiterea valorii
;   din tabel, la o iesire D/A a sistemului.
=====
;
;   Definirea adreselor porturilor sistemului:
;   -----
PORT0      EQU    0H      ; adresa convertorului A/D. Operatiile de
;                          ; output la PORT0 determina startul conversiei
;                          ; A/D; input de la portul PORT0 permite citirea
;                          ; valorii convertite de catre convertor
PORT1      EQU    1H      ; adresa convertorului D/A. Prin output la PORT1,
;                          ; se transmite data ce trebuie convertita;
;
;-----
; Zona de cod a programului ;
;-----
;
;   la adresa 0, salt la programul principal
;
;           ORG 0 ; directiva pentru asamblor; se genereaza cod de la adresa 0
;
;           JMP PRINC ; instructiune de salt la programul principal
;
;   la adresa 4, salt la subrutina de tratare a intreruperii
;
;           ORG 4
;
;           JMP INTRER
;
;-----
;   programul principal, fie de la adresa 10H
;
;           ORG 10H
;
PRINC: MOV CS, 280H      ; initializare CS; in CS se incarca adresa
;                          ; varfului stivei initiale a programului, fie 280H
;           OUT (PORT0), AL ; start conversie A/D
;                          ; (nu conteaza continutul acumulatorului)
;           EI             ; activare intreruperi
BUCLA:
;           HLT            ; procesorul in HALT, pe perioada cat nu se
;                          ; trateaza intreruperi de la CA/D

```

```

JP BUCLA          ; la revenire din intreruperi, salt din nou la
                  ; instructiunea de HALT
; -----
; Urmeaza subrutina de tratare a intreruperilor;
; aici sare programul (prin executarea instructiunii de salt,
; de la adresa 0004H, la aparitia intreruperii, la sfarsitul conversiei).
; Inainte de saltul la rutina de intrerupere, se salveaza in stiva continutul PC.
; La executia instructiunii de RET, programul se reia de unde fusese intrerupt,
; prin refacerea PC din stiva.
;
INTRER:
; La intrarea in rutina de tratare a intreruperilor, se salveaza in mod explicit,
; prin program, in stiva, toate registrele microprocesorului, al caror continut este
; modificat de catre rutina respectiva. La iesirea din subrutina, se reface, in
; ordine inversa celei de salvare, continutul initial al acestor registre.
; Pentru exemplul dat, sunt salvate registrele AX si SI.
;
PUSH AX           ; salvare registru AX in stiva
PUSH SI           ; salvare registru SI in stiva
;
IN AL, PORT0      ; citire in A valoare convertita de CD/A
MOV SI, [TABEL]   ; SI contine adresa primului element al
                  ; tabloului cu valorile functiei
MOV AH, 0         ; AX = AH & AL, contine valoarea conversiei
                  ; A/D, identica cu indexul elementului de tablou,
                  ; corespunzator transformarii functionale
ADD AX, SI        ; se aduna la SI, valoarea din AX, si se obtine
                  ; in AX rezultatul, adresa elementului cautat
                  ; in tabelul TABEL
MOV SI, AX        ; SI contine adresa elementului cautat in tabel
MOV AL, [SI]      ; in AL se aduce, prin citire din memorie,
                  ; de la adresa denotata de SI, valoarea functiei
;
OUT PORT1, AL     ; valoarea functiei este transmisa convertorului D/A
OUT PORT0, AL     ; start o noua conversie A/D
; cind se va termina conversia, convertorul A/D va genera o noua intrerupere
; refacerea continutului registrelor SI si AX
POP SI
POP AX
EI               ; activare intreruperi (dezactivate automat la
                  ; intrarea in rutina de intreruperi)
RET              ; revenire la programul intrerupt (se reface PC
                  ; din stiva, salvat la acceptarea intreruperii)
; -----
; in continuare, urmeaza tabelul cu valori ale functiei (fie, pe cate un octet)
; se utilizeaza directiva DB --- define byte
; lista contine, despartite de virgule, elementele tabloului (vor fi in PROM),
; valori memorate in ordinea scrierii:
;
TABEL:
DB 0, 1, 5, 17, 22, 9
DB 8, 7, 11, 33, 19,...
... (in total 256 numere, cu valori intre 0 si 255)
...

```

...

END

; directiva indicand asamblorului sfarsitul textului sursa

3.7.3 Concluzii

Prezentul paragraf nu a urmarit si nici nu poate sa acopere problemele ridicate de implementarea reala a *hardware*-ului si *software*-ului unui sistem microprocesor in totalitatea sa. Cu toate acestea, au fost prezentate elementele fundamentale si problemele ce apar in acest sens.

Cateva dintre problemele care nu au fost, cu buna stiinta, abordate, sunt urmatoarele:

- *nu s-au facut consideratii asupra timpilor de acces si selectie* pentru diferitele circuite utilizate (presupuse deci ca opereaza cu viteze satisfacatoare relativ la viteza microprocesorului);
- *nu s-au facut consideratii privitoare la capacitatea de incarcare (fan-out)* a magistralelor si semnalelor de iesire ale microprocesorului (de regula insuficiente si necesitand circuite repetoare cu fan-out mai mare);
- *nu s-au detaliat circuitele de selectie* ale memoriei si porturilor (circuite combinationalale cu grad relativ redus de complexitate);
- *s-au facut ipoteze simplificatoare* privind semnalele de control, tratarea intreruperii (salt la adresa fixa) --- apare doar la unele tipuri de microprocesoare;
- *nu s-a evaluat durata executiei subrutinei de intrerupere*; compararea acesteia cu durata conversiei *A/N* permite, in cazul unei durate mai mici a executiei subrutinei, sa se execute instructiunea *OUT PORT0*, de start a conversiei, imediat dupa citirea valorii convertite. (Astfel se castiga timp, executandu-se instructiuni de prelucrare a valorii citite --- cautare in tabel, *OUTPUT* valoarea functiei --- in timp ce se converteste deja o noua valoare). Mai trebuie mentionat din nou ca, dispunand eventual si de un circuit de ceas programabil, care sa permita achizitia de date la intervale fixe de timp, structura *hardware* a sistemului prezentat ar putea implementa, cu un alt program, de exemplu un regulator de orice tip (in functie de algoritmul utilizat, de parametrii implementati), bineinteles pentru un proces cu constante de timp corelate cu durata maxima a executiei calculelor care ar impune temporizarea ceasului.

4. MICROPROCESORUL INTEL 8086

La patru ani de la prezentarea primului microprocesor pe 8 biti — *8080* (magistrala de adrese de 16 biti, magistrala de date de 8 biti), ca urmare a succesului de piata repurtat cu acesta si a dezvoltarii tehnologiei H-MOS, firma *INTEL* realizeaza microprocesorul *8086* (magistrala multiplexata — 20 linii de adresa, 16 linii de date) cu care se implanteaza solid si in domeniul microprocesoarelor de 16 biti.

Patru concepte arhitecturale noi la momentul respectiv au stat la baza proiectarii si realizarii familiei lui *8086*. Primul este acela al **segmentarii memoriei**, prin care se permite programarea modulara, deschizindu-se astfel calea implementarii tehnicilor de protectie si partajare a memoriei. Al doilea este acela al posibilitatii implementarii cu usurinta a unor **moduri de adresare caracteristice limbajelor de programare de nivel inalt** (bazata, indexata, pentru structuri de date). Un alt concept atins il constituie cresterea puterii de calcul pe baza unui **set de registri optimizat** pentru a realiza functii generale si specializate de procesare a datelor, precum si introducerea unitatilor aritmetice de calcul de tipul coprocesoarelor aritmetice, echipamente care se conecteaza pe magistralele microprocesorului in paralel cu acesta, crescind mult viteza de lucru a sistemului in unele aplicatii. In fine, microprocesorul *8086* dispune de un set de instructiuni codificat anume pentru utilizarea la viteza crescuta si cu **eficienta maxima a memoriei**.

8086 extinde performantele unui microprocesor la niveluri neatinse de predecesorul sau pe 8 biti: operatii aritmetice pe 16 biti cu numere cu si fara semn (inclusiv inmultiri si impartiri); operatii complexe de manipulare a sirurilor de caractere precum si la nivel de bit; posibilitatea realizarii de programe relocatabile in mod dinamic; adresarea directa a unui megabyte de memorie; posibilitatea configurarii unor structuri foarte variate din punct de vedere al complexitatii sistemului; simplitatea implementarii unor sisteme complexe multi-microprocesor.

Realizarea acestor performante cu un circuit in capsula cu doar 40 de pini (ca si *8080*) se indeplineste pe doua cai si vom semnala inca din acest moment al prezentarii cele doua caracteristici hardware esentiale ce definesc functionarea microprocesorului *I 8086*. Prima caracteristica o constituie **multiplexarea in timp a magistralei de adrese si date** (descrisa in detaliu in sectiunea 4.4.2.). In al doilea rind, microprocesorul are o **configuratie interna comutabila** pentru adaptarea la nivelul de complexitate a sistemului in care este utilizat. Astfel, in sistemele simple, *8086* isi genereaza si controleaza singur semnalele de pe magistrala de control; in sistemele complexe, magistrala de control este generata de un circuit specializat din familia lui *8086*, circuit denumit "controler de magistrala" (*8288 Bus Controller*), opt dintre conexiunile fizice ale microprocesorului fiind comutate pentru a

indeplini functiunile de coordonare necesare. Un singur pin al lui 8086, pinul 33 ($\overline{MN} / \overline{MX}$), conectat la masa sau la alimentare, comanda comutarea configuratiei interne a microprocesorului pentru adaptarea la sistemul extern.

4.1 ARHITECTURA PROCESORULUI

Asa cum s-a prezentat in capitolul precedent, executia unui program intr-un sistem cu microprocesor are ca efect (intr-o abordare simplificata) repetarea ciclica a pasilor de mai jos:

1. Extragerea urmatoarei instructiuni din memorie.
2. Citirea unui operand (daca instructiunea o cere).
3. Executia instructiunii.
4. Scrierea rezultatului (daca instructiunea o cere).

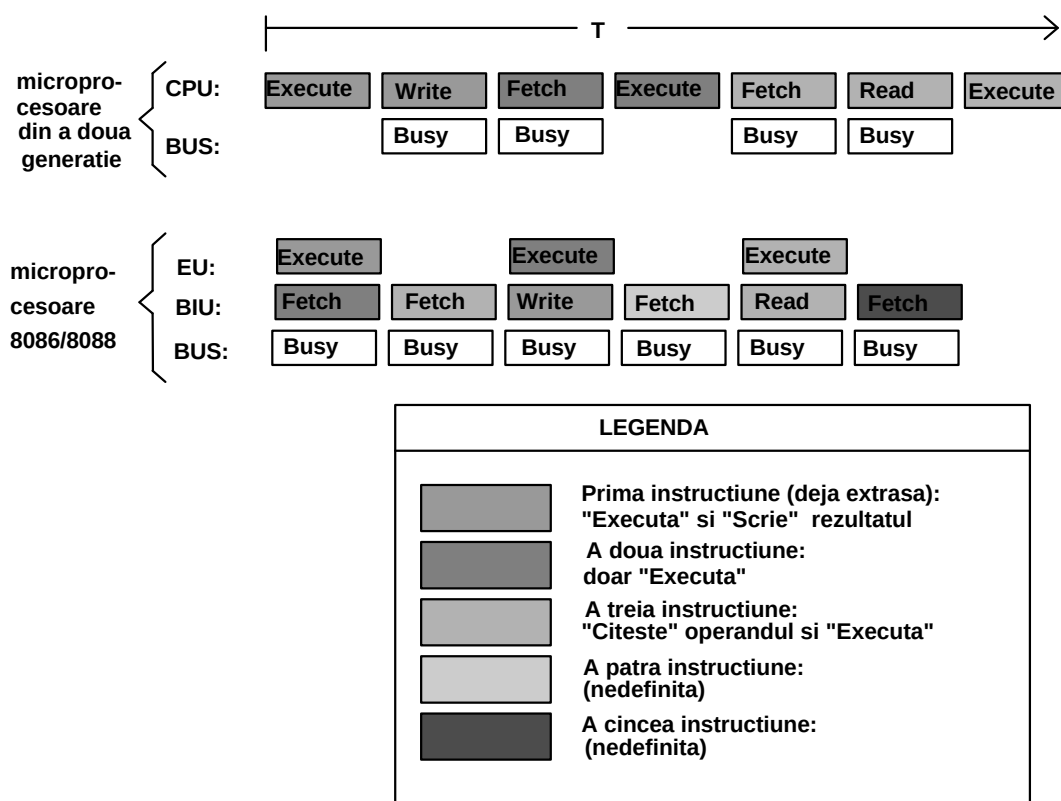


Fig. 4.1. Suprapunerea in timp a etapelor de extragere si executie a instructiunilor procesorului 8086

Datorita arhitecturii specifice, executia acestor pasi are loc in doua unitati separate de procesare a datelor din cadrul CPU: EU (*execution unit*) — unitatea de executie — si BIU (*bus interface unit*) — unitatea de interfata cu magistrala. In EU are loc executia instructiunilor in timp ce BIU extrage instructiunile, citește operanzii si scrie rezultatele in memorie. Cele doua unitati pot opera independent una de cealalta si pot asigura — in majoritatea cazurilor — suprapunerea in timp a etapei de extragere a unei instructiuni cu etapa de executie a unei instructiuni precedent extrase din memorie. In acest mod, practic "dispare"

timpul necesar extragerii instructiunilor din memorie, crescand viteza de lucru a microprocesorului deoarece EU executa instructiuni al caror cod a fost deja adus de catre BIU din memorie in microprocesor. Figura 4.1 ilustreaza aceasta suprapunere in timp a fazelor de extragere si executie a instructiunilor printr-un exemplu in care timpul necesar executiei a trei instructiuni este mai mic la 8086 in comparatie cu microprocesoarele din generatia precedenta, doua instructiuni in plus fiind deja extrase si asteptand sa fie executate.

4.1.1 EU — Unitatea de executie

Unitatea de executie contine registrii de uz general, unitatea aritmetico-logica, registrul indicatorilor de conditie, un bloc logic de control si o magistrala interna de date de 16 biti (figura 4.2).

Funcțiile EU acopera executia tuturor instructiunilor, furnizarea datelor si a adreselor catre BIU, controlul registrelor de uz general si al indicatorilor de conditie. Cu exceptia citorva pini de control, unitatea de executie este complet izolata de "lumea exterioara". Asa cum se evidentiaza in figura 4.2, EU preia instructiunea urmatoare de executat dintr-o coada de asteptare alimentata continuu de unitatea de interfata cu magistrala.

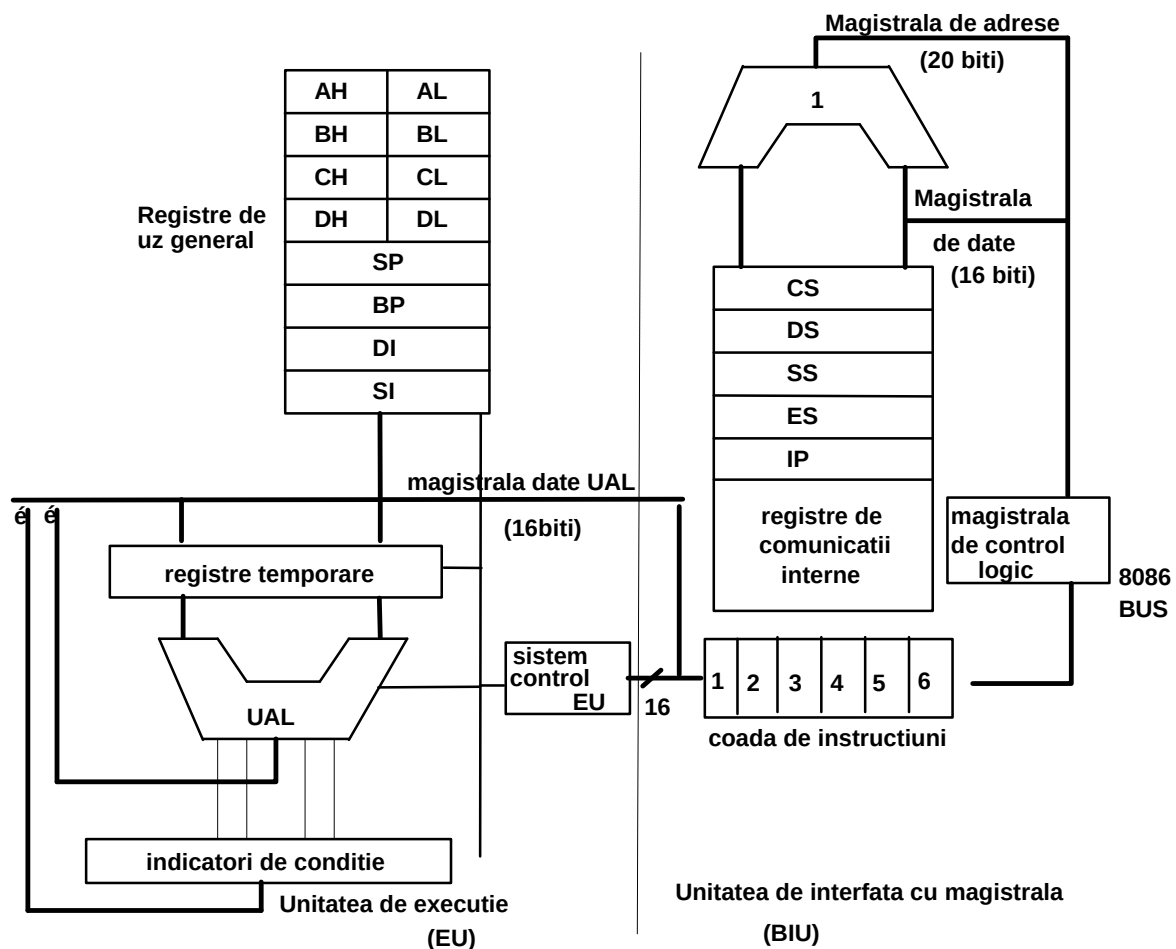


Fig. 4.2. 8086 — schema bloc

În situația în care nu există nici o instrucțiune de preluat pentru a fi executată, unitatea de execuție așteaptă până când coada este realimentată de către BIU. Dacă în cursul execuției instrucțiunii este necesar accesul la o locație de memorie sau la un echipament periferic, EU solicită unității de interfață cu magistrală să transfere date, executând ciclul de magistrală corespunzător (citire/scriere la memorie sau port). Totodată, deși magistrala EU are doar 16 biți, se poate accesa în exterior întregul spațiu de un megaoctet de memorie prin intermediul BIU care asigură relocarea adresei înainte de fiecare transfer.

4.1.2 BIU — Unitatea de interfață cu magistrală

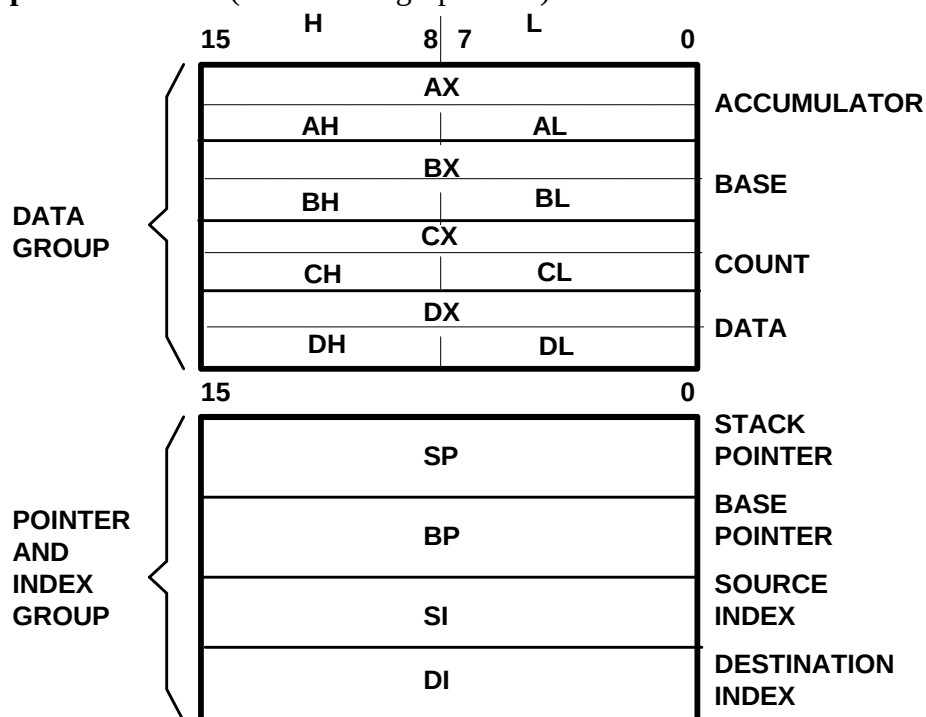
Unitatea de interfață cu magistrală execută toate operațiile externe de magistrală ce sunt necesare pe parcursul extragerii și execuției unei instrucțiuni. Ea se compune din registrul de segment, un registru de tip contor de program denumit **pointer de instrucțiuni** (*instruction pointer*), registri de comunicație internă, o schemă logică pentru generarea adresei pe cele 20 de linii de adresă ale microprocesorului 8086 și pentru controlul magistralei multiplexate precum și o **coadă de instrucțiuni** (*queue*). Aceasta este realizată cu o memorie RAM de 6 octeți și conține instrucțiuni care sunt extrase în avans de BIU și urmează să fie preluate de EU pentru decodificare și execuție.

Cele două unități de procesare ale CPU operează independent una de alta — în sensul că, ori de câte ori doi sau mai mulți octeți din coadă sunt liberi, iar EU nu solicită BIU la efectuarea vreunui ciclu de magistrală, unitatea de interfață cu magistrală execută în avans cicluri de extragere de instrucțiuni pentru a realimenta locațiile libere din coada de instrucțiuni. Acest mod de lucru permite BIU să furnizeze EU instrucțiuni extrase anterior fără a monopoliza magistrala sistemului căci, în mod normal, în majoritatea situațiilor coada de instrucțiuni conține cel puțin un octet ce poate fi preluat de EU pentru decodificare și execuție. În plus, cum sistemele cu microprocesor 8086 au uzual magistrală de date de 16 biți, într-un singur ciclu de extragere se alimentează coada cu doi octeți, cu excepția cazurilor când adresa de la care se citește instrucțiunea următoare este impară, așa cum se va arăta ulterior.

Instrucțiunile extrase în avans de BIU sunt cele care urmează în mod logic într-o procesare serială a programului, ele aflându-se în memorie în locații adiacente și la adrese superioare adresei instrucțiunii care se execută la un moment dat. În cazul în care EU execută o instrucțiune care transferă controlul programului către o altă locație de memorie, BIU resetează coada, extrage instrucțiunea de la noua adresă, transferând-o imediat unității de execuție, apoi începe realimentarea cozii de la noua locație. De asemenea, BIU suspendă operațiile de extragere de instrucțiuni (cu excepția celei în curs de desfășurare) ori de câte ori unitatea de execuție solicită efectuarea pe magistrală a unui transfer cu memoria sau cu un port de intrare/ieșire.

4.1.3 Registrele de uz general

Microprocesorul 8086 are 8 registre generale de 16 biti grupate in doua seturi a cite 4 registre fiecare: **registrele de date** (uneori denumite grupul registrelor H&L de la *high* si *low*) si **registrele pointer si index** (denumite si grupul P&I).



Un registru apartinand grupului H&L se caracterizeaza prin faptul ca poate fi adresat ca registru de 16 biti, dar se compune din doua entitati de 8 biti, partea *high* si partea *low*, care pot fi adresate la rindul lor separat, ca registre de 8 biti. Registrele pointer si index nu pot fi adresate decat ca registre de 16 biti.

Atat registrele de date cat si registrele pointer si index pot fi folosite in majoritatea operatiilor aritmetice si logice, oricare dintre ele putand juca rolul registrului "acumulator" existent la generatiile precedente de microprocesoare. Pentru a permite utilizarea unui set compact, dar puternic de instructiuni, anumite registre sunt folosite in mod implicit de unele instructiuni, asa cum arata tabelul 4.1.

Tabelul 4.1. Utilizarea implicita a registrelor de uz general

REGISTRU	OPERATII
AX	Inmultiri, impartiri si I/E pe cuvint
AL	Inmultiri, impartiri si I/E pe octet, translatari, aritmetica zecimala
AH	Inmultiri si impartiri pe octet
BX	Translatari
CX	Operatii cu siruri de caractere, contor pentru operatii repetate
CL	Deplasari si rotiri cu mai mult de o pozitie
DX	Inmultiri si impartiri pe cuvint, I/E cu adresare indirecta
SP	Operatii cu stiva

sI, DI	Operatii cu siruri de caractere
--------	---------------------------------

4.1.4 Registrele de segment

Spatiul fizic de memorie de 1 Mb direct adresabil de microprocesorul 8086 este divizat in segmente logice de pana la 64 kb fiecare (vezi sectiunea 4.2.1.). CPU are acces direct in orice moment la patru segmente logice ale caror adrese de baza (adresele de inceput ale segmentelor) se afla in registrii de segment ai microprocesorului.

15	0
CS	CODE SEGMENT
DS	DATA SEGMENT
SS	STACK SEGMENT
ES	EXTRA SEGMENT

Registrul segmentului de cod (**CS**) contine adresa de inceput a segmentului din care sunt extrase instructiunile — segmentul de cod. Stiva programului se afla in asa numitul segment de stiva catre care "pointeaza" registrul segmentului de stiva (**SS**). Mai exista de asemenea doua segmente de date, unul propriu-zis (**DS**) si unul suplimentar (**ES**) fiecareia fiindu-i asociat cate un registru de segment ce contine adresa de inceput

respectiva. Si registrele de segment sunt accesibile programatorului, continutul lor putind fi modificat de anumite instructiuni.

4.1.5 Registrul pointerului de instructiuni

Registrul **pointerului de instructiuni** (*Instruction Pointer*) este similar registrului **contor de program** (*Program Counter*) al microprocesoarelor pe 8 biti. Actualizat de catre BIU, el contine **offsetul** (distanta in octeti) urmatoarei instructiuni, masurat de la inceputul segmentului curent de cod. Astfel spus, IP reprezinta in mod normal un pointer catre urmatoarea instructiune ce urmeaza a fi *extrasa* de catre BIU, iar atunci cand este salvat in stiva se modifica automat pentru a indica offsetul urmatoarei instructiuni ce urmeaza a fi *executata* de EU. Operarea cu continutul IP odata salvat in stiva constituie calea prin care acesta poate fi modificat indirect in decursul executiei unui program.

4.1.6 Indicatorii de conditie

Microprocesorul 8086 are 6 biti de **stare** si 3 biti de **control** grupati in **registrul indicatorilor de conditie** (*flags*). Cei de stare sunt pozitionati de unitatea de executie pentru a reflecta anumite proprietati ale rezultatului unei operatii aritmetice sau logice. Acestia pot fi utilizati de un grup al setului de instructiuni pentru a modifica secventialitatea executiei programului in functie de rezultatul operatiei anterioare. In general, indicatorii starii programului reflecta urmatoarele conditii:

- AF** **Auxiliary Carry Flag.** Daca $AF=1$, a existat un transport dinspre bitul 7 spre bitul 8 sau un imprumut dinspre bitul 8 catre bitul 7. Acest indicator este folosit indeosebi in cazul instructiunilor ce implica operatii aritmetice cu numere zecimale codificate binar.
- CF** **Carry Flag.** Daca $CF=1$, a existat un transport dinspre sau un imprumut catre cel mai semnificativ bit (*MSB*) al rezultatului reprezentat pe 8 sau 16 biti. Acest indicator este utilizat de instructiunile ce implica operatii de adunare sau scadere cu numere reprezentate pe unul sau mai multi octeti. Biti ai operanzilor din memorie sau registri pot fi izolati in CF prin intermediul instructiunilor de rotire si deplasare.
- OF** **Overflow Flag.** Daca $OF=1$, a aparut o depasire aritmetica, adica s-a pierdut cel mai semnificativ bit al rezultatului datorita faptului ca dimensiunea acestuia a depasit capacitatea de reprezentare a locatiei destinatie. Este de remarcat faptul ca exista o instructiune (*INTerrupt On Overflow*) care genereaza o cerere de intrerupere pentru semnalarea aparitiei acestei situatii.
- SF** **Sign Flag.** $SF=1$ indica faptul ca cel mai semnificativ bit al rezultatului are valoarea 1. Cum numerele intregi cu semn sunt reprezentate in *complement fata de 2*, rezulta ca SF arata semnul rezultatului (0=pozitiv, 1=negativ).
- PF** **Parity Flag.** $PF=1$ atunci cand rezultatul are un numar par de biti pozitionati pe 1. Poate fi utilizat pentru verificarea erorilor de transmisie a datelor.
- ZF** **Zero Flag.** Valoarea zero a rezultatului este evidentiata prin $ZF=1$.

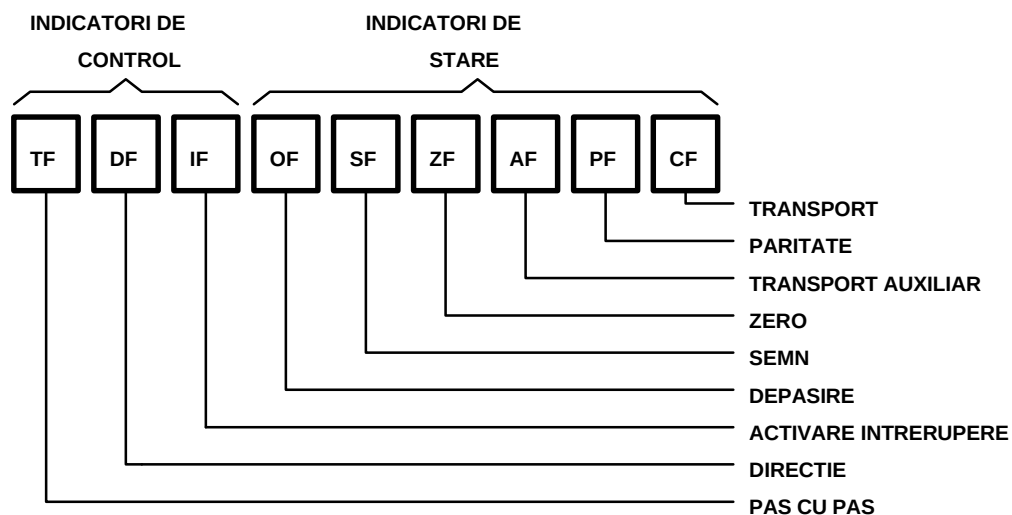


Fig. 4.3. Indicatorii de conditii

Cei 3 indicatori de control pot fi modificati prin program pentru a determina anumite operatii ale procesorului. Astfel:

DF

Direction Flag. Acest indicator este utilizat de instructiunile ce opereaza cu siruri de caractere (*string*) si semnalizeaza autodecrementarea (DF=1) respectiv autoincrementarea (DF=0) registrilor SI si DI care contin ofsetul adreselor sursa si destinatie ce intervin in transfer.

IF

Interrupt — enable Flag. Setarea IF prin instructiunea *SeT Interrupt — enable flag* permite CPU sa recunoasca cererile de intrerupere externa mascabile, in timp ce resetarea aceluiasi indicator cu instructiunea *CLear Interrupt — enable flag* le va dezactiva. Modificarea valorii lui IF nu are efect asupra intreruperilor generate intern in CPU sau a celor externe nemascabile.

TF

Trap Flag. Daca TF=1, procesorul intra in modul de **operare pas cu pas** in care CPU genereaza automat o intrerupere interna dupa fiecare instructiune pentru a permite examinarea starii programului si deci depanarea acestuia.

4.2 ORGANIZAREA MEMORIEI

Microprocesorul 8086 poate adresa direct un spatiu de $2^{20} = 1\text{Mo} = 1.048.576$ octeti organizat liniar cu adresele joase la inceput si adresele inalte la sfirsit. Instructiunile si datele pe octet sau cuvânt pot fi plasate liber la orice adresa, fie ea para sau impara, pentru a permite o stocare densa a codului programului in memorie — acesta fiind unul din conceptele impuse la proiectarea procesorului, asa cum s-a aratat in paragraful introductiv (v. si fig. 4.4). Totusi, trebuie remarcat faptul ca memorarea unei variabile de tip *word* la o adresa impara — situatie in care se spune ca variabila este **nealiniata** (*unaligned*) — anuleaza din punct de vedere al transferului variabilei respective avantajul microprocesorului 8086 de a avea o magistrala de date de 16 biti (vezi sectiunea 4.4.5.). In ceea ce priveste instructiunile, alinierea sau nealinierea lor nu afecteaza performantele microprocesorului datorita cozii de asteptare din BIU.

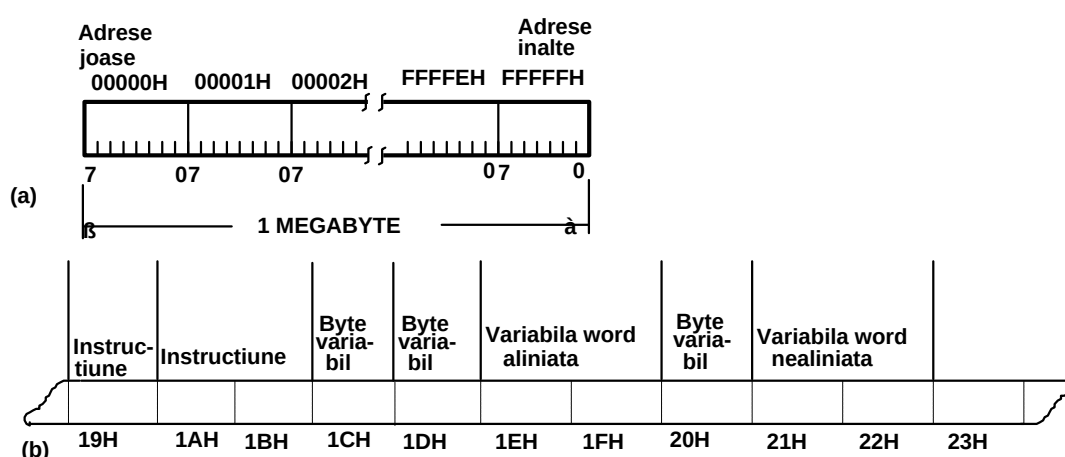


Fig. 4.4. Organizarea memoriei — Memorarea variabilelor si instructiunilor

Conform convențiilor *INTEL*, o variabilă de tip *word* este întotdeauna memorată cu octetul cel mai semnificativ în locația de memorie cu adresă superioară.

Această regulă se extinde și la memorarea variabilelor de tip *pointer* (variabile pe un cuvânt dublu folosite pentru a adresa date în afara segmentului de date sau de program ce este adresabil în mod curent la momentul respectiv): cuvântul ce conține offsetul se memorează la cele două adrese inferioare, cuvântul ce conține adresă de bază a segmentului se memorează la cele două adrese superioare la care este stocat *pointerul*, iar în cadrul fiecărui cuvânt, octetul cel mai semnificativ este la rândul lui memorat la adresa superioară (vezi figura 4.5).

246H				247H				4H				5H				6H				7H			
0	4	8	C	HEXA				6	5	0	0	4	C	3	B	HEXA							
0000	0100	1000	1100	BINAR				0110	0101	0000	0000	0100	1100	0011	1011	BINAR							

Valoarea variabilei *word* memorată la 246H este 8C04H
 VALOAREA POINTERULUI STOCAT LA 4H: ADRESA BAZA SEGMENT:3B40H OFFSET:65H

Fig. 4.5. Memorarea variabilelor de tip *word* și *pointer*

4.2.1 Segmentarea memoriei

8086 localizează spațiul de memorie fizică de 1 Mo prin intermediul unui grup de **segmente logice** definite de fiecare aplicație în parte. Segmentele sunt unități logice de memorie continuă cu mărimea de maxim 64 ko fiecare, independente unele de altele și adresabile în mod separat. Fiecărui segment *i* se asignează software o **adresă de bază** ce reprezintă adresa de început a segmentului respectiv, mai concret — adresa **celei mai de jos** locații de memorie a segmentului.

Singura restricție ce se aplică segmentelor logice este ca această adresă de bază să fie un multiplu de 16. În rest, segmentele pot fi adiacente, disjuncte, suprapuse parțial sau total (vezi figura 4.6), astfel încât o locație fizică de memorie poate fie să nu aparțină nici unui segment, fie să aparțină unui singur segment, fie mai multor segmente.

Amintindu-ne că în BIU există 4 registre de segment ce conțin fiecare câte o adresă de bază de segment, rezultă că la un moment dat există 4 **segmente adresabile în mod curent** (figura 4.7). În cursul execuției unui program, accesul la instrucțiuni sau date aparținând altor segmente decât cele adresabile în mod curent se face prin simplă modificare a conținutului registrelor de segment corespunzătoare.

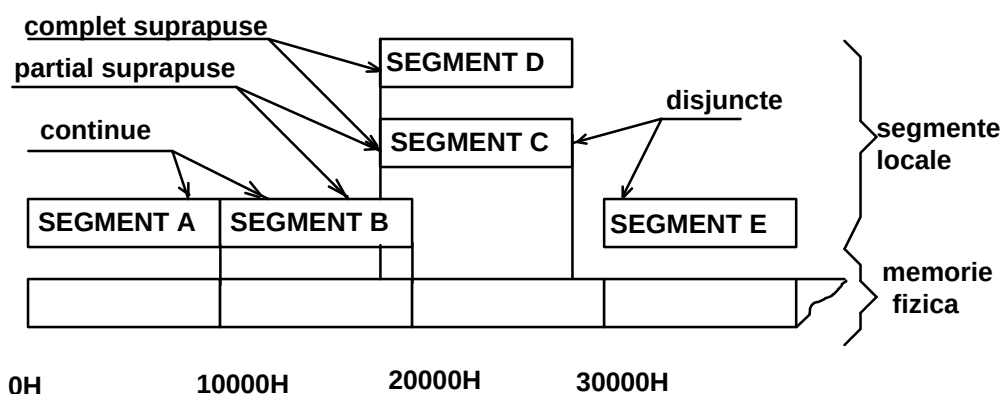


Fig. 4.6. Alocarea segmentelor in memoria fizica

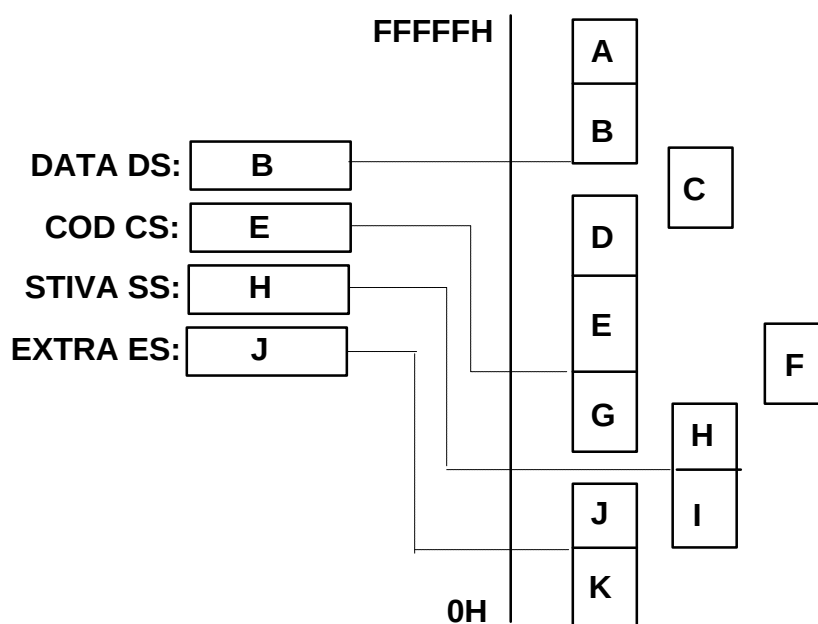


Fig. 4.7. Segmentele adresabile in mod curent

Fiecare aplicatie trebuie sa-si defineasca de la inceput adresele de baza ale segmentelor pentru a se asigura localizarea informatiilor in memorie. Daca 64 ko de cod, 64 ko de stiva si 128 ko de date sunt suficienti pentru aplicatia respectiva, registrele de segment raman nemodificate pana la terminarea executiei acesteia. Pentru aplicatiile mai complexe, se recomanda fie modularea programelor, fie utilizarea corecta a registrelor de segment.

4.2.2 Generarea adresei fizice

Se poate imagina ca fiecare locatie de memorie are doua feluri de adresa: **fizica** si **logica**. Adresa fizica este acea valoare pe 20 de biti care identifica in mod unic fiecare octet din spatiul de memorie de 1Mo. Ea este cuprinsa intre 0H si FFFFFH si exista pe liniile magistralei multiplexate la inceputul fiecarui ciclu de scriere sau citire in/din memorie.

Programatorul operează însă mai mult cu adresa logică decât cea fizică, ceea ce permite scrierea unui program fără a cunoaște dinainte locul unde codul acestuia va fi încărcat în memorie și facilitează controlul dinamic al resurselor de memorie. O adresă logică constă dintr-o valoare de bază a segmentului și o valoare de offset, ambele fiind mărimi fără semn reprezentate pe 16 biți. Pentru orice locație de memorie, valoarea de bază a segmentului indică primul octet conținut în segment (începutul segmentului) iar valoarea de offset reprezintă distanța în octeți de la acest început până la locația respectivă. Primul octet (cel mai de jos) aparținând unui segment are deci offsetul egal cu zero. Având în vedere conceptul de segment logic, rezultă că o locație fizică de memorie poate avea o multime de adrese logice. Acest fapt este ilustrat în figura 4.8 în care locația de memorie fizică 2C3H este conținută în două segmente logice suprapuse parțial, unul începând la 2B0H iar altul la 2C0H.

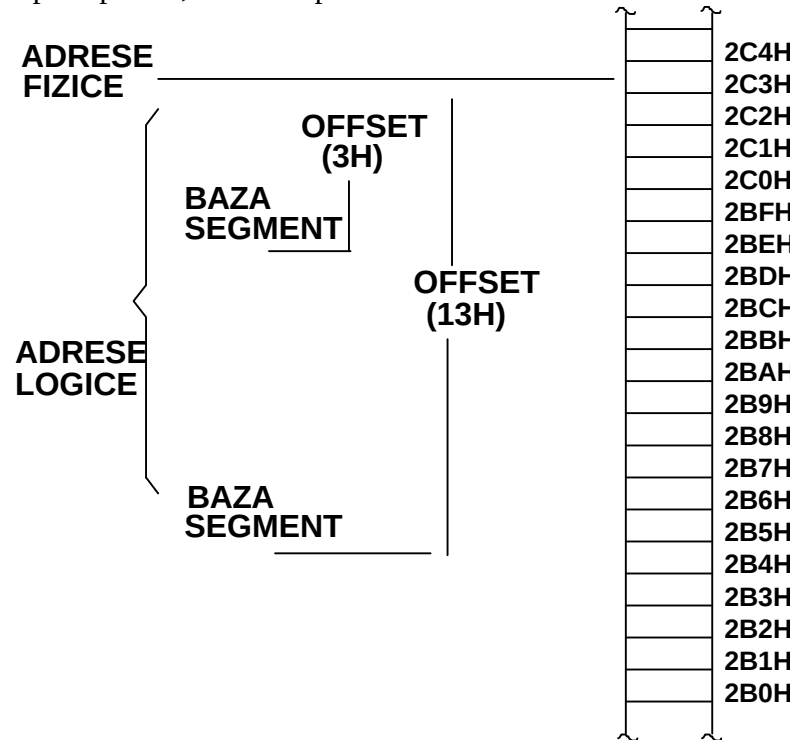
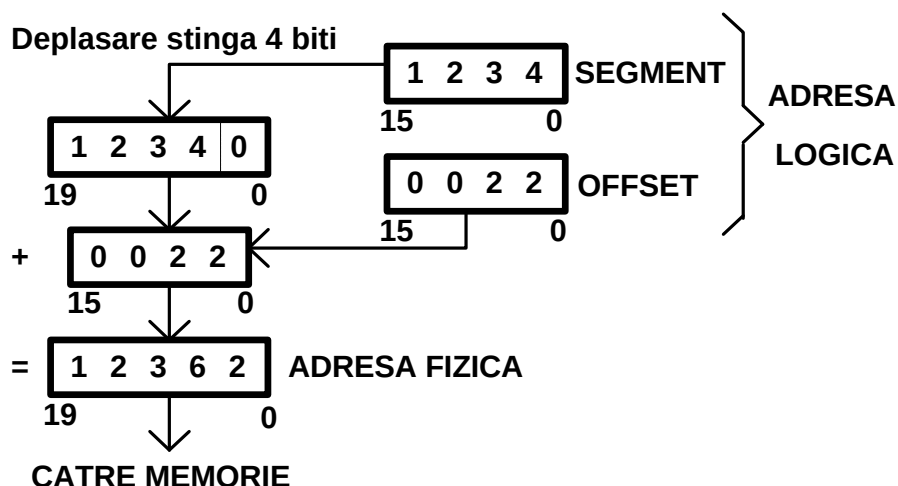


Fig. 4.8. Adresa fizică și adrese logice

Ori de câte ori unitatea de interfață cu magistrala accesează memoria — pentru a extrage o instrucțiune sau pentru a obține sau stoca o variabilă — ea generează adresa fizică pe baza adresei logice. Aceasta se realizează prin deplasarea la stânga cu patru poziții binare a valorii de bază din registrul de segment și adunarea valorii de offset.



BIU obtine adresa logica a locatiei de memorie din diferite surse, in functie de scopul accesului la memorie (vezi tabelul 4.2).

Instructiunile sunt extrase din segmentul de cod iar ofsetul este dat de registrul IP, operatiile cu stiva sunt executate in segmentul de stiva curent iar ofsetul este dat de registrul SP, s.a.m.d. Ofsetul unei variabile din memorie este calculat de unitatea de executie in functie de modul de adresare specificat in instructiune; rezultatul se numeste **adresa efectiva** (*effective address — EA*).

In majoritatea cazurilor, utilizarea segmentelor implicite pentru accesul datelor in memorie este convenabila pentru programator. Este posibil insa si accesul unei variabile in oricare dintre segmentele adresabile in mod curent (cu exceptia instructiunilor ce manipuleaza siruri de caractere unde operandul destinatie este obligatoriu sa se gaseasca in segmentul de date suplimentar). Pentru a obtine acest lucru, instructiunea trebuie precedata de un prefix — *segment override prefix* — care indica BIU ce registru de segment sa utilizeze pentru a accesa variabila respectiva.

Tabelul 4.2. Sursele de adresa logica

SCOPUL ACCESULUI LA MEMORIE	SEGMENT IMPLICIT	SEGMENT ALTERNATIV	OFSET
Extragere instructiune	CS	-	IP
Operatie cu stiva	SS	-	SP
Variabila (cu exceptia urmatoarelor)	DS	CS, ES, SS	Adresa Efectiva
Sursa sir caractere	DS	CS, ES, SS	SI
Destinatie sir caractere	ES	-	DI
BP utilizat ca registru de baza	SS	CS, DS, ES	Adresa Efectiva

Modul de obtinere a adresei fizice si structura segmentata a memoriei creaza posibilitatea realizarii de programe ce nu depind de locul unde sunt incarcate in memorie, adica programe

relocatabile in mod dinamic, ceea ce permite utilizarea deplina a memoriei disponibile si realizarea de sisteme **multitasking**. Astfel, programele inactive pot fi indepartate din memorie (transferate pe disc) si spatiul ocupat de ele alocat altor programe. Activarea unui program se face prin incarcarea lui in orice zona libera din memorie si lansarea in executie. In mod similar, daca un program are nevoie de un spatiu de memorare continuu de capacitate mare, iar zona disponibila este fragmentata, segmentele altui program pot fi compactate pentru a elibera spatiul, situatie ilustrata in figura 4.9.

Pentru a fi relocabil dinamic, un program trebuie sa nu-si incarce sau modifice registrele de segment sau sa transfere controlul intr-o locatie aflata in afara segmentului curent de cod. Toate offseturile vor fi relative la valorile fixe continute in registrele de segment. Programul poate fi mutat oriunde in memorie, prin simpla actualizare a adreselor de baza de segment.

4.2.3 Stiva

Stiva microprocesorului 8086 este implementata in memorie prin intermediul registrului de segment SS si al registrului indicator de stiva SP (*stack pointer*).

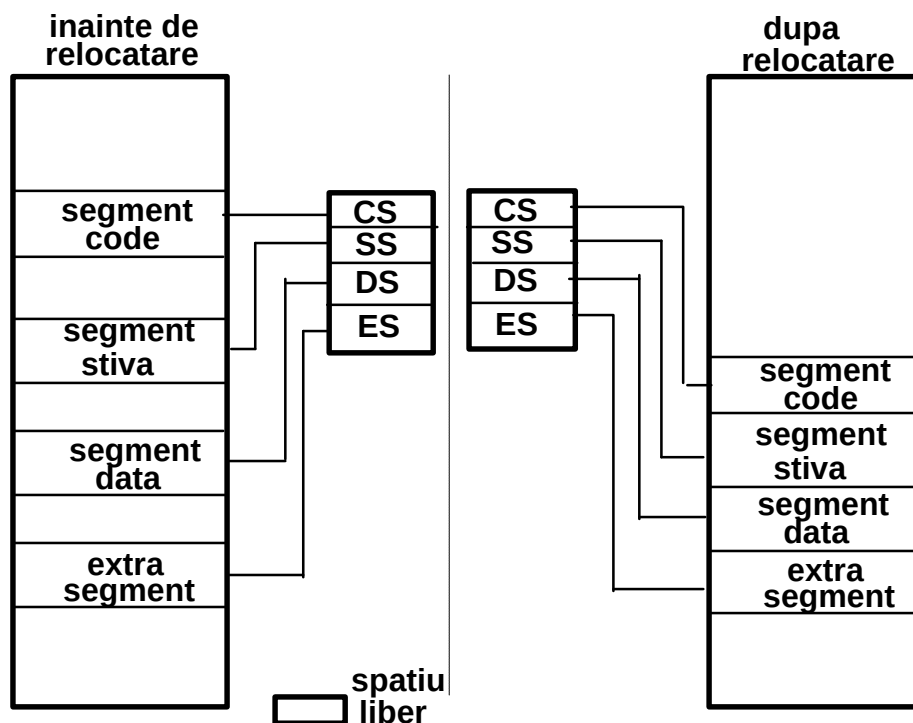


Fig. 4.9. Relocatarea dinamica a programelor

Un sistem poate avea un numar nelimitat de stive a cite cel mult 64 ko fiecare, dar una singura este direct adresabila la un moment dat: aceasta este stiva curenta, sau pur si simplu "stiva". SP contine offsetul varfului stivei (*TOS — top of stack*) fata de adresa de baza indicata de registrul SS. Cum elementele stivei sunt cuvinte de 16 biti, instructiunile care opereaza cu stiva cauzeaza decrementarea (pentru *PUSH*) respectiv incrementarea (pentru *POP*) cu 2 a

registrului indicator de stiva SP (vezi figura 4.10). Cu alte cuvinte, stiva **creste in jos** in memorie, catre adresa de baza a segmentului de stiva. Este de remarcat faptul ca operatiile cu stiva nu transfera elemente dintr-o zona in alta a acesteia, nici nu le sterg, ci doar provoaca modificarea varfului stivei prin actualizarea registrului SP.

O constrangere ce se aplica tuturor segmentelor de memorie, nu numai stivei, este cauzata de existenta unor locatii de memorie dedicate sau rezervate. Aceste locatii, prezentate in figura 4.11, sunt: 0H-7FH (128 octeti) si FFFF0H-FFFFFH (16 octeti). Ele sunt folosite pentru intreruperi si pentru pornirea sistemului dupa reset, putand fi si rezervate de firma pentru dezvoltari ulterioare, iar utilizarea lor de catre programator trebuie sa se faca respectind scopul pentru care au fost definite ca "locatii speciale".

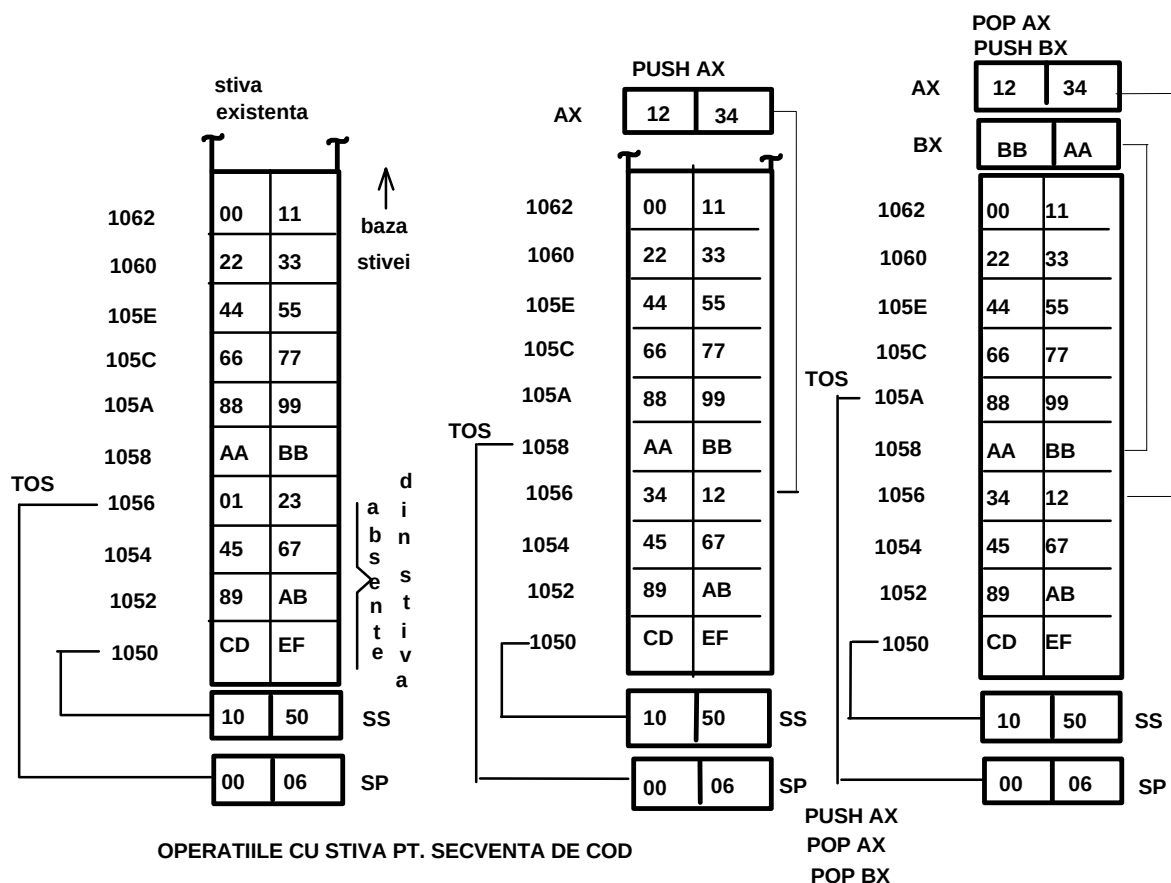


Fig. 4.10. Operatii cu stiva

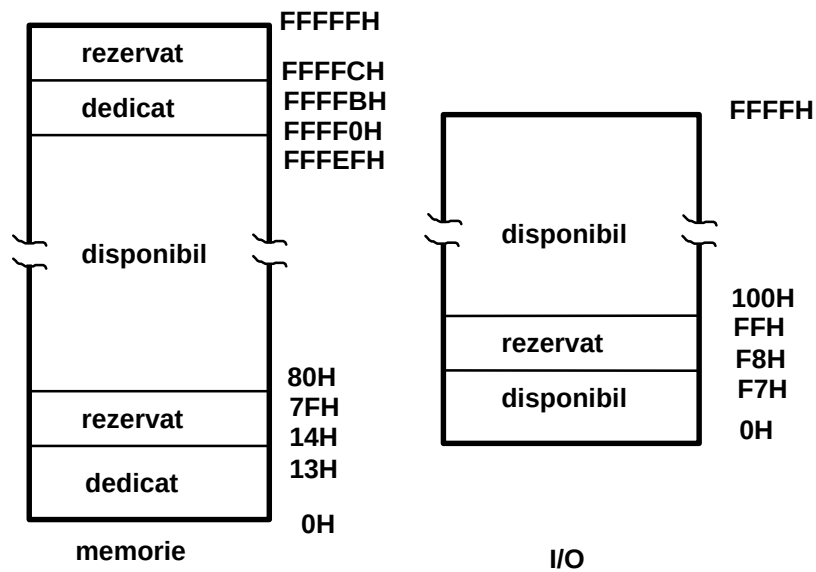


Fig. 4.11. Spatiul de memorie si I/E cu precizarea "locatiilor speciale"

4.3 ORGANIZAREA SPATIULUI DE INTRARE/IESIRE

Microprocesorul 8086 dispune de un spatiu larg de intrare/iesire separat de memorie, precum si de instructiuni care transfera date intre CPU si echipamente localizate in acest spatiu. Acestea pot de asemenea sa fie dispuse in zona de memorie, pentru a beneficia de o putere sporita a setului de instructiuni si a modurilor de adresare, si in ceea ce priveste operatiile de intrare/iesire.

4.3.1 Spatiul de intrare/iesire propriu-zis

Spatiul propriu-zis este de 64 ko porturi de 8 biti sau 32 ko porturi de 16 biti. Instructiunile *INput* si *OUTput* transfera date intre acumulator (AL pentru transfer pe octet si AX pentru transfer pe cuvint) si porturile localizate in spatiul de intrare/iesire.

Spre deosebire de memorie, spatiul de I/E nu este segmentat: pentru a accesa un port, unitatea de interfata cu magistrala plaseaza adresa portului (cuprinsa, deci, intre 0H si FFFFH) pe cele mai putin semnificative 16 linii ale magistralei de adrese. In functie de forma instructiunii de I/E, adresa respectiva poate fi specificata ca o valoare fixa in instructiune, sau ca o variabila luata din registrul DX.

4.3.2 Porturile de I/E plasate in spatiul de memorie

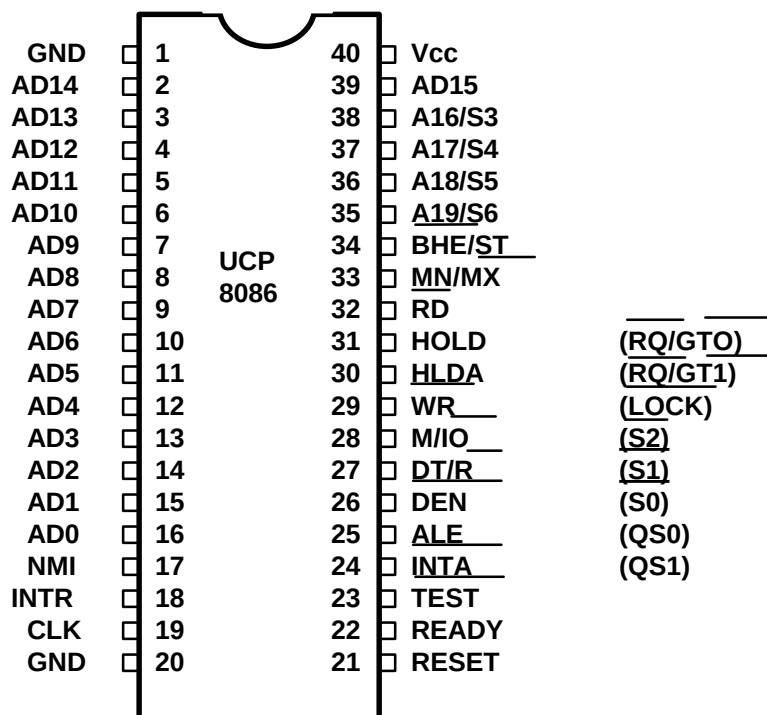


Fig. 4.12. Microprocesorul 8086

Si in sistemele cu microprocesor 8086 se poate plasa zona de I/E peste spatiul de memorie (*memory-mapped I/O*) — daca, bineinteles, acesta nu este complet ocupat cu circuite de memorie. Aceasta permite efectuarea operatiilor de I/E prin instructiuni de lucru cu memoria, microprocesorul accesand locatii in spatiul de 1 Mo de memorie, locatii care in realitate sunt porturi de I/E. Avantajul este ca se dispune de o clasa puternica de instructiuni si de moduri de adresare caracteristice memoriei, ceea ce asigura o mare flexibilitate programarii operatiilor de I/E. Pretul care trebuie platit pentru aceasta este, pe de o parte faptul ca se indisponibilizeaza o parte a spatiului de memorie (desi 1 Mo inseamna o dimensiune mare a memoriei), iar pe de alta parte, instructiunile cu memoria se executa intr-un timp mai mare si sunt mai putin compacte decat simplele *IN* si *OUT*.

4.4 SPECIFICATII HARDWARE

4.4.1 Definirea pinilor

Microprocesorul 8086 are o magistrala de date si adrese **multiplexata in timp** (la inceputul ciclului masina informatia prezenta pe magistrala multiplexata reprezinta o adresa, ulterior ea schimbându-se si reprezentând data implicata in transferul efectuat la adresa respectiva) ceea ce permite unui numar de pini sa indeplineasca functii duale si in consecinta microprocesorului sa poata fi incorporat intr-o singura capsula cu 40 de pini.

Deasemenea, asa cum se va arata ulterior, un numar de pini de control ai CPU sunt definiti diferit in functie de cum este conectat un singur pin de intrare, pinul 33 ($MN / \overline{MX}$), la masa sau la V_{cc} . In figura 4.12 este prezentata asignarea pinilor, denumirile in paranteza corespunzand conectarii pinului 33 la masa, iar in tabelul 4.3, functiile si tipul semnalelor.

Tabelul 4.3. Functiile si tipul semnalelor microprocesorului 8086

SEMNALE COMUNE		
Nume	Funcție	Tip
AD15-AD0	Address/Data Bus — Magistrala de adrese/date	Bidirect., 3-state
A19/S6-A16/S3	Address/Status — Adrese/Stare	Iesire, 3-state
BHE/S7	Bus High Enable/Status — Activare octet superior pe mag./Stare	Iesire, 3-state
MN/MX	Minimum/Maximum Mode Control — Control mod min./max.	Intrare
RD	Read Control — Control citire	Iesire, 3-state
TEST	Wait On Test Control — Test asteptare	Intrare
READY	Wait State Control — Sincronizare transfer pe mag. de date	Intrare
RESET	System Reset — Initializare sistem	Intrare
NMI	Non-Maskable Interrupt Request — Cerere intrerupere nemascabila	Intrare
INTR	Interrupt Request — Cerere intrerupere mascabila	Intrare
CLK	System Clock — Ceas de sistem	Intrare
Vcc	+ 5 V	Intrare
GND	Ground — Masa	
SEMNALE PENTRU MODUL MINIM ($MN/MX = V_{cc}$)		
Nume	Funcție	Tip
HOLD	Hold Request — Cerere magistrala	Intrare
HLDA	Hold Acknowledge — Acceptare cerere magistrala	Iesire
WR	Write — Control scriere	Iesire, 3-state
M/IO	Memory/IO Control — Control selectie memorie/port de I/E	Iesire, 3-state
DT/R	Data Transmit/Receive — Control sens transfer date pe magistrala	Iesire, 3-state
DEN	Data Enable — Activare date pe magistrala	Iesire, 3-state
ALE	Address Latch Enable — Activare adrese pe magistrala	Iesire
INTA	Interrupt Acknowledge — Acceptare cerere de intrerupere	Iesire
SEMNALE PENTRU MODUL MAXIM ($MN/MX = GND$)		
Nume	Funcție	Tip

RQ/GT1,0	Request/Grant Bus Access Control — Cerere/Acceptare acces mag.	Bidirectional
LOCK	Bus Priority Lock Control — Control acces pe magistrala	Iesire, 3-state
S2 — S0	Bus Cycle Status — Specificare stare ciclu de magistrala	Iesire, 3-state
QS1, QS0	Instruction Queue Status — Specificare stare coada de instructiuni	Iesire

4.4.2 Functionarea magistralei multiplexate

Pentru a intelege modul de operare pe magistrala multiplexata in timp, trebuie analizat ciclul de magistrala al BIU. Sa observam ca, in esenta, un **ciclu de magistrala** este un eveniment asincron care incepe prin aparitia adresei unui port de I/E sau a unei locatii de memorie, urmata fie de un semnal de control de citire (pentru a captura sau "a citi" data de la echipamentul accesat), fie de un semnal de control de scriere impreuna cu data asociata (pentru a transmite sau "a scrie" data in echipamentul adresat). La rindul lui, echipamentul selectat — memorie sau port — accepta data de pe magistrala pe durata ciclului de scriere sau plaseaza data ceruta pe magistrala in timpul ciclului de citire. La terminarea ciclului, echipamentul respectiv memoreaza data care a fost scrisa (uzual prin intermediul unor circuite de tip *latch* pe intrare), sau indeparteaza de pe magistrala data citita de microprocesor (trecandu-si circuitele tampon de iesire pe magistrala in stare de inalta impedanta).

Asa cum se arata in figura 4.13, toate ciclurile de magistrala constau din cel putin patru perioade de ceas sau *T-states* identificate ca T1, T2, T3 si T4. CPU plaseaza pe magistrala adresa locatiei de memorie sau portului cu care doreste sa efectueze un transfer pe durata starii T1. In cazul unui ciclu de scriere, data este plasata pe magistrala de CPU din starea T2 pana in starea T4. In cazul unui ciclu de citire, CPU accepta data prezenta pe magistrala pe perioada starilor T3 si T4, iar magistrala multiplexata de adrese/date este in stare flotanta in T2 pentru a permite CPU sa faca trecerea din modul de scriere (iesirea adreselor) in modul de citire (intrarea datelor).

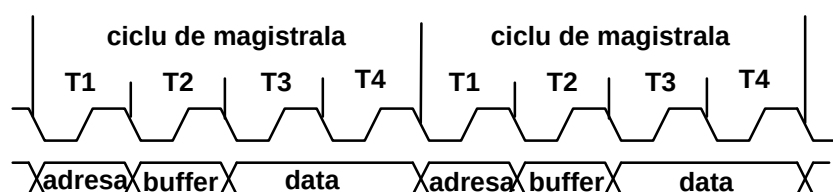


Fig. 4.13. Ciclu tipic de magistrala

Sa ne amintim insa faptul ca un ciclu de magistrala (deci un acces in afara microprocesorului) are loc doar atunci cand este cerut de EU pentru executia unei instructiuni sau cand BIU trebuie sa realimenteze coada de instructiuni. Prin urmare, intre ciclurile de magistrala ce corespund acestor situatii vor exista perioade de ceas in care magistrala este neutilizata de microprocesor. Aceste perioade de ceas de inactivitate a microprocesorului pe magistrala se numesc *idle states* — T_I . Desi ele pot apare ca urmare a diferitor situatii (chiar si acceptarea

cererii de magistrala a unui coprocesor cauzeaza aceasta comportare), cel mai frecvent caz este acela al executiei unor instructiuni "lungi". Astfel, in figura 4.14 este prezentata executia unei instructiuni de inmultire a doua registre de 8 biti — **MULTiply** — care dureaza intre 70 si 77 cicluri de ceas. Intre ciclurile normale de magistrala de extragere de instructiuni de catre BIU, apar perioade de inactivitate pe magistrala, T_I , in timpul carora in microprocesor se desfasoara intern executia instructiunii.

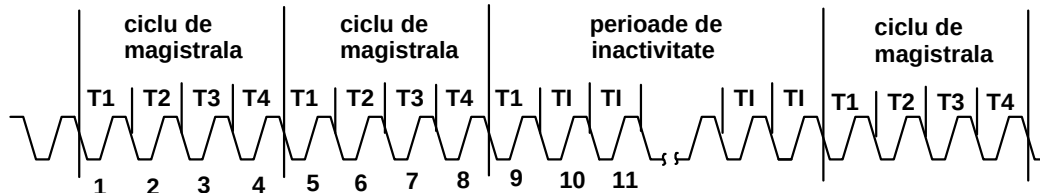
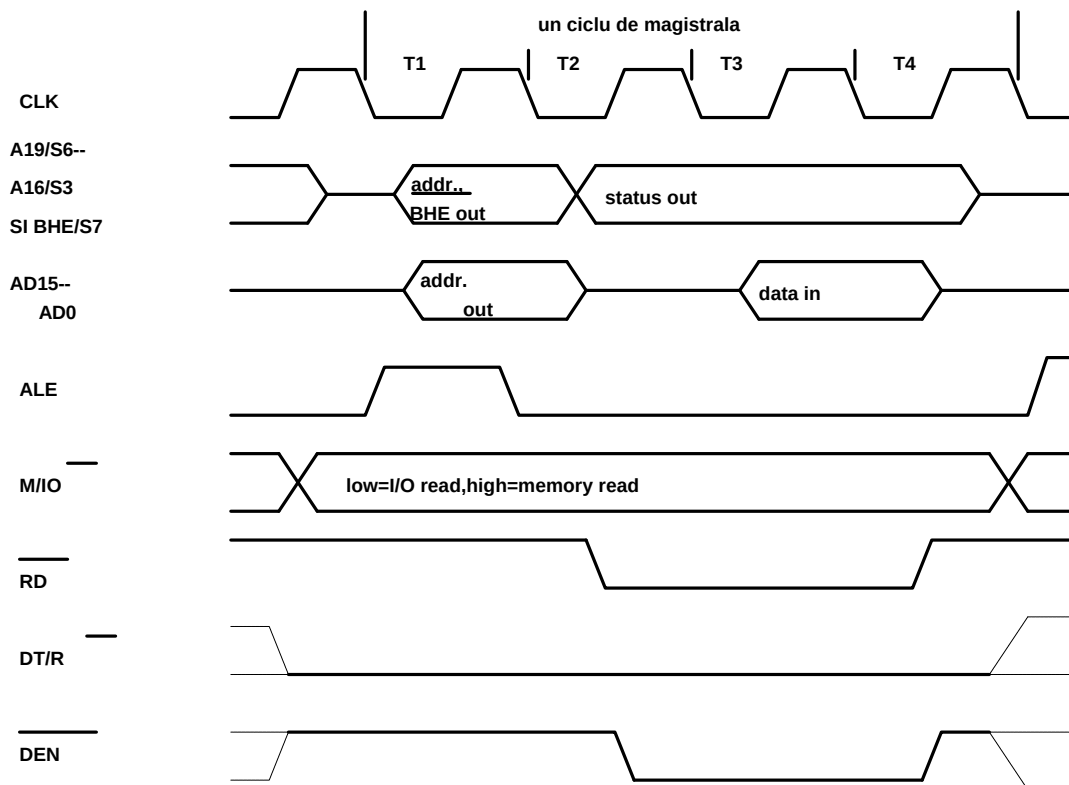


Fig. 4.14. Idle states pe magistrala

In plus fata de starile de inactivitate pe magistrala deja descrise, CPU dispune de un mecanism care insereaza stari aditionale intr-un ciclu de magistrala pentru a sincroniza transferul cu acele circuite de memorie sau port care sunt mai lente si nu pot lucra la viteza microprocesorului. Aceste stari T_W suplimentare sunt numite **stari de asteptare** — T_W — wait



si sunt inserate, daca este cazul, intre starile T3 si T4. Pe durata unei stari T_W informatia ramane neschimbata pe magistrala, iar in momentul in care circuitul selectat poate efectua transferul, el semnaleaza CPU sa iasa din starea de asteptare si sa intre in starea T4.

In diagramele de timp ce urmeaza sunt prezentate ciclurile de magistrala de citire si scriere in cazul in care microprocesorul 8086 este configurat in modul minim ($MN / \overline{MX}$ conectat la

+5V, microprocesorul controland direct toate semnalele de pe magistrala de control a sistemului).

Referindu-ne la figurile 4.15 si 4.16, sa observam ca 8086 plaseaza o adresa de 20 de biti pe magistrala multiplexata pe durata starii T1. Pe durata T2, aceasta adresa este indepartata de pe magistrala pentru a permite trecerea celor mai putin semnificative 16 linii fie in stare de inalta impedanta pentru efectuarea unei operatii de citire, fie in stare de iesire a datei furnizate in cazul unei operatii de scriere. In acelasi timp, cele 4 linii "de jos" ale magistralei isi schimba semnificatia din linii de adresa (AD19-AD16) in linii de stare¹ (S6-S3) a ciclului de magistrala. Aceste linii se mentin si pe perioada T3 in care pe cele 16 linii "de jos" fie este prezenta in continuare data scrisa, fie este esantionata (*strobată*) data citita. Daca nu se solicita de catre echipamentul selectat inserarea unei stari T_W , dupa T3 urmeaza T4 in care ciclul de magistrala se termina, liniile de control sunt dezactivate si magistrala trece in stare de inalta impedanta.

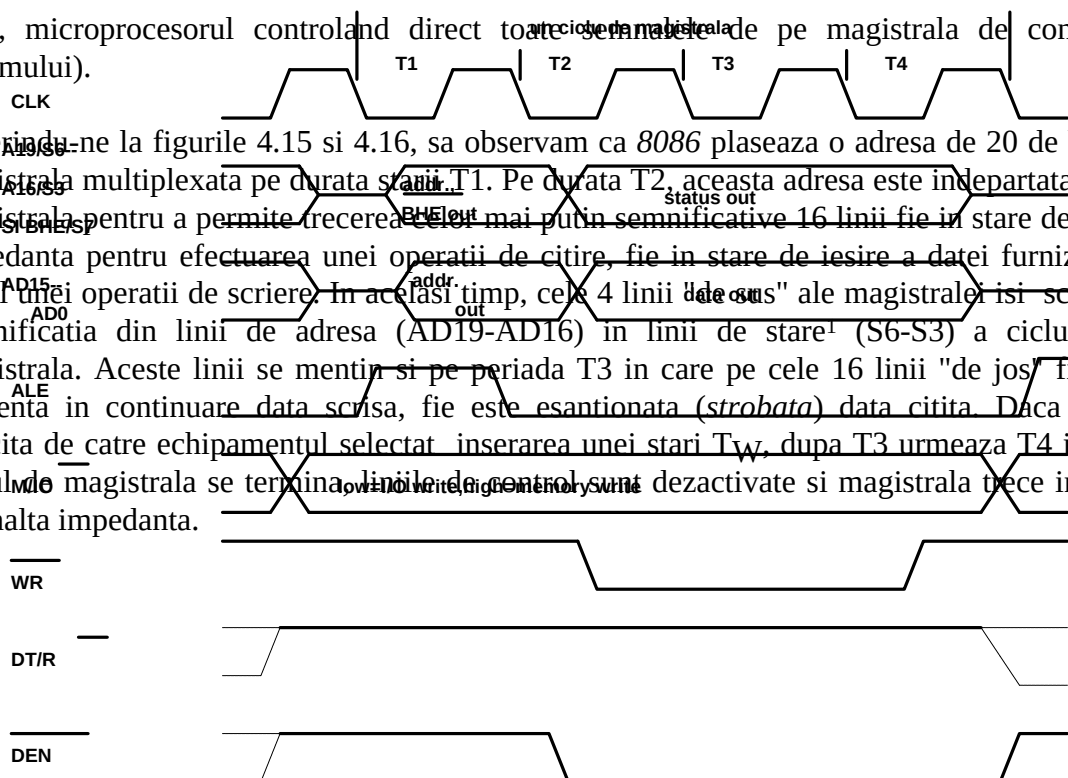


Fig. 4.16 Operatia de scriere — ciclu de magistrala

Fig. 4.15. Operatia de citire — ciclu de magistrala

¹ — S3 si S4 codifica in care din cele 4 segmente de memorie este localizata data ce intervine in transfer;
— S5 reflecta starea indicatorului de conditie IF;
— S6 = 0 indica faptul ca 8086 este pe magistrala;

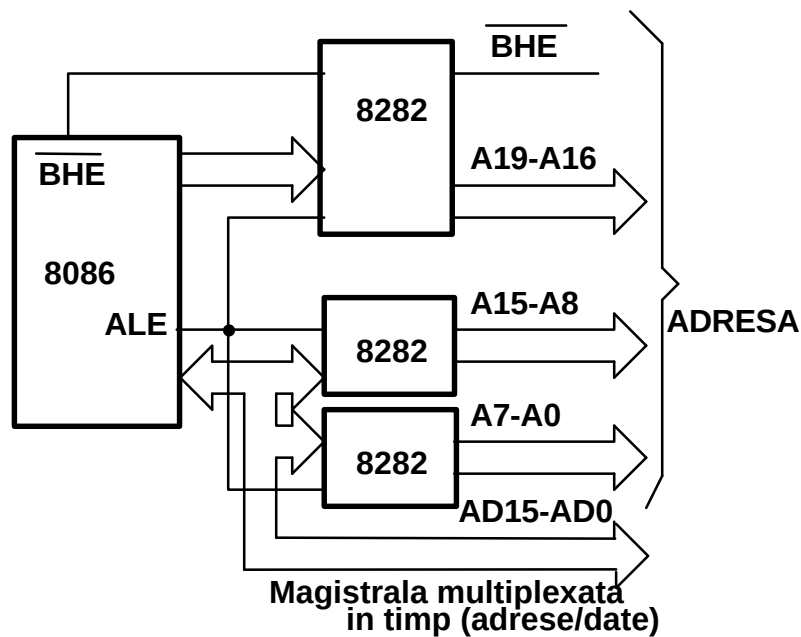


Fig. 4.17. Magistrala de date utilizata direct, multiplexata in timp.

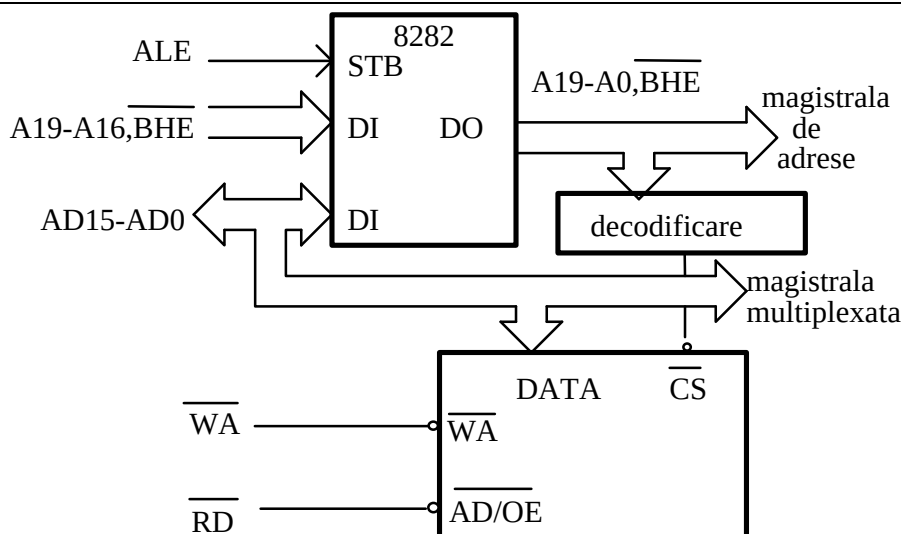
Cum majoritatea circuitelor de memorie sau port dintr-un sistem microprocesor au nevoie de o adresa stabila pe durata ciclului de magistrala, este necesar ca magistrala de adrese/date a lui 8086 sa fie demultiplexata, generandu-se magistrale separate de adrese si de date. Pentru aceasta, microprocesorul furnizeaza semnale specifice de control, in afara celor de citire — $\overline{RD}$, de scriere — $\overline{WR}$ si de selectie a memoriei sau a porturilor de I/E — $M/\overline{IO}$.

Astfel, pe durata starii T1 a fiecarui ciclu de magistrala, deci atunci cand pe magistrala multiplexata se afla informatia de adresa, microprocesorul (sau in modul maxim, un controler de magistrala — 8288) isi activeaza semnalul ALE al carui front cazator care va fi folosit in sistem pentru a determina memorarea acestei adrese in niste circuite de tip *latch* (de exemplu circuitele *latch* pe 8 biti neinversoare 8282, respectiv inversoare 8283). Aceste circuite, beneficiind si de un *fan-out*¹ sporit, vor distribui in sistem ceea ce reprezinta o magistrala de adrese stabila pe durata intregului ciclu si care se numeste simplu, magistrala de adrese a sistemului.

Magistrala de date nu poate fi demultiplexata datorita comportarii diferite in timp intr-o operatie de scriere fata de una de citire, precum si datorita timpilor diferiti de raspuns al echipamentelor sistemului la primirea unui semnal de citire. In consecinta, magistrala de date multiplexata poate sa fie folosita direct asa cum este, adica multiplexata, sau poate sa fie **bufferata** (prevazuta cu circuite tampon bidirectionale conectate la liniile AD15-AD0).

Prima situatie este ilustrata in figura 4.17. Daca un echipament de memorie sau un port de intrare/iesire este conectat direct la magistrala multiplexata, proiectantul trebuie sa se asigure ca echipamentul respectiv nu intervine pe magistrala pe durata starii T1 pentru a distruge

¹ capacitatea unei iesiri de a controla un anumit numar de intrari de circuit



cumva adresa furnizata in acel interval (ceea ce ar insemna de fapt arderea unor circuite prin punerea in scurtcircuit a doua iesiri care pot fi pe nivele de tensiune diferita).

Pentru a evita acest lucru, *driver*-ele de iesire ale echipamentului nu trebuie sa fie activate de semnalul de selectie de circuit — *Chip Select* — ci de un semnal de activare a iesirii — *Output Enable* — care sa fie activ dupa trecerea lui T1 si sa semnifice necesitatea citirii datei pe magistrala, adica semnalul de control citire — *Read*. Multe circuite de memorie sau port dispun de o intrare de tip $\overline{OE}$, astfel incat se pot conecta direct pe magistrala de date multiplexata ca in figura 4.18. (daca circuitul nu dispune de o astfel de intrare, atunci semnalul de selectie de circuit — $\overline{CS}$ — va trebui generat doar in prezenta unui semnal activ de citire sau scriere date pe magistrala).

Atat datorita simplitatii modului de interfatare a circuitelor pe magistrala cat si necesitatii unui *fan-out* corespunzator, uzual se utilizeaza magistrala de date buferata prin intermediul unor circuite tampon bidirectionale (exemplu, *transceiver*-ele pe 8 biti 8286 — neinversoare sau 8287 — inversoare). Specializate pentru acest mod de utilizare, circuitele tampon respective sunt comandate de doua semnale de control ale CPU : $\overline{DEN}$ — semnifica intervalul de timp cand se efectueaza un transfer de date pe magistrala, cu alte cuvinte este un semnal de activare a magistralei de date, respectiv $DT / \overline{R}$ — semnal ce controleaza sensul transferului datei (1=scriere, 0=citire). In acest mod se obtine si magistrala de date a sistemului (figura 4.19).

4.4.3 Circuitul de ceas

Pentru a controla sincronizarea semnalelor pe magistrala, CPU are nevoie de un semnal extern de ceas. Circuitul de ceas *INTEL 8284*, apartinand familiei *8086*, in afara de generarea semnalelor de ceas, dispune si de posibilitatea interfatarii hardware cu un circuit de initializare (*reset*) a microprocesorului precum si de un mecanism pentru inserarea de stari de asteptare — T_W — in cadrul ciclului de magistrala.

8284 necesita conectarea la intrare din exterior a unui cristal de cuarț de frecvență tripla față de frecvența la care funcționează *8086* caruia îi furnizează direct semnalul de ceas *CLK*. In plus față de

Fig. 4.18. Conectarea circuitelor direct pe magistrala multiplexata plus ieseirea *CLK*,

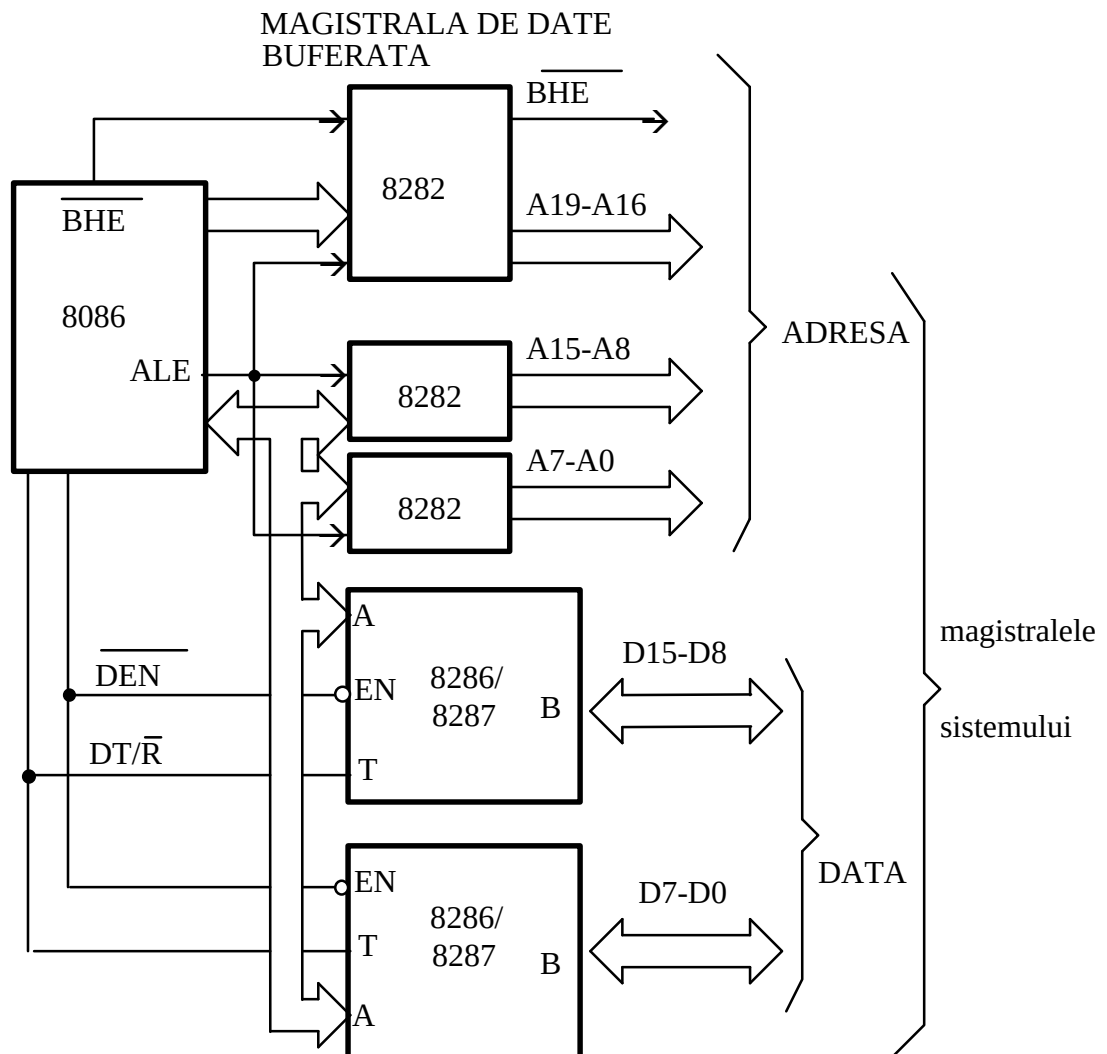
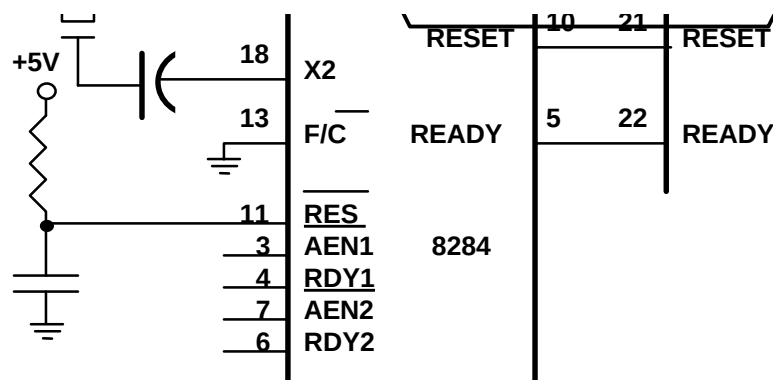
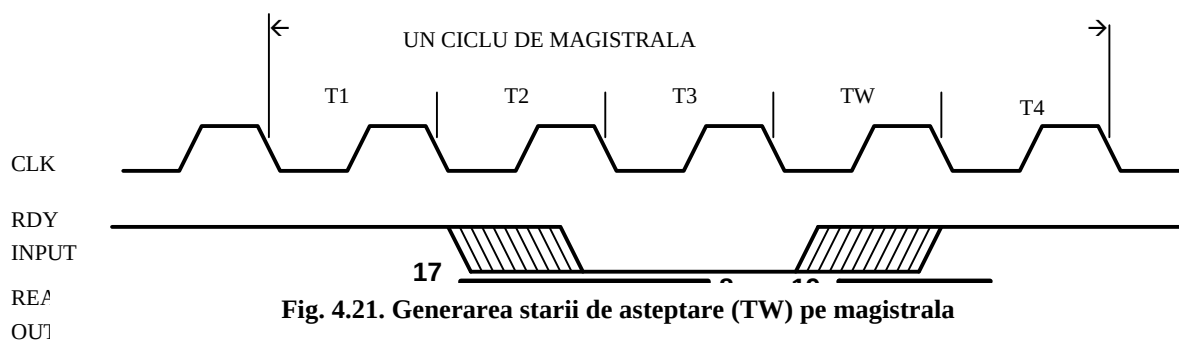


Fig. 4.19. Magistrala de date buferată

sunt disponibile si un semnal **Peripheral CLoCK** de frecventa jumatare (fata de CLK) si un semnal TTL — **OSCillator** avind aceeasi frecventa ca si cristalul de quart. Aceste semnale suplimentare de ceas pot fi utilizate de echipamentele din sistem.

Funcția de *hardware reset* a lui 8284 este implementată cu un circuit trigger Schmitt activat de intrarea $\overline{RES}$. Când această intrare este trasă în zero de un contact la masă, se activează semnalul **RESET**, furnizat de 8284 microprocesorului sincronizat cu semnalul de ceas **CLK**. Semnalul de **RESET** trebuie să fie activ cel puțin patru perioade de ceas pentru a provoca initializarea procesorului și ca atare extragerea și executia primei instrucțiuni de program care se încarcă din memorie de la locația FFFF0H. Uzual, un circuit extern RC conectat la intrarea **RES** realizează funcția de initializare la pornirea tensiunii de alimentare (*power-on reset*) dacă semnalul furnizat circuitului de ceas durează cel puțin 50 de microsecunde. Iesirea de **RESET** ce se aplică microprocesorului se utilizează și în restul sistemului, ca un semnal general de initializare a sistemului.

Inserarea de stări de așteptare în ciclul de magistrală al CPU se realizează prin dezactivarea uneia din cele două intrări de *ready* ale 8284 (**ReaDY1** sau **ReaDY2**) de către echipamentul



care dorește încetinirea transferului pe magistrală. Fiecare dintre intrările **RDY1** și **RDY2** poate participa la generarea semnalului **READY** către microprocesor, după cum intrarea

Fig. 4.20. Interfatarea circuitului de ceas 8284 la microprocesorul 8086

corespondentă $\overline{AEN1}$ sau $\overline{AEN2}$ (*address enable*) este activă.

Iesirea **READY**, care este sincronizată cu semnalul de ceas **CLK**, este conectată direct la intrarea **READY** a lui 8086. Așa cum se arată în figura 4.21, când echipamentul selectat este

lent si se doreste inserarea de stari de asteptare pe magistrala, el trebuie sa dezactiveze intrarea RDY a lui 8284 inainte de sfarsitul starii T2 astfel incat iesirea *READY* sa fie inactiva la sfarsitul starii amintite. Starea de asteptare obtinuta este inserata intre T3 si T4. Pentru a se iese din T_W , echipamentul selectat va activa intrarea RDY a lui 8284 care la randul lui va activa intrarea *READY* in 8086 la sfarsitul starii *wait*, ceea ce va permite CPU sa intre in starea T4 pentru a termina ciclul de magistrala. Evident ca, prin mentinerea in continuare a lui RDY pe nivel *low*, se pot insera multiple stari T_W inainte de T4.

4.4.4 Modul minim/maxim. Controlerul de magistrala 8288

Tabelul 4.4.

PIN	MINIM	MAXIM
31	HOLD	RQ/GT0
30	HLDA	RQ/GT1
29	WR	LOCK
28	M/IO	S2
27	DT/R	S1
27	DEN	S0
25	ALE	QS0
24	INTA	QS1

Probabil ca una din facilitatile cele mai deosebite oferite de 8086 este posibilitatea de a selecta hardware configuratia de baza a masinii prin simpla conectare la masa sau alimentare a pinului 33 ($MN / \overline{MX}$), prin care se defineste modul de lucru. Practic se realizeaza astfel o selectie intre doua definitii functionale ale unui subset de iesiri ale lui 8086. Asignarea pinilor in cele doua moduri de functionare este prezentata in tabelul 4.4.

MODUL MINIM

Pentru a configura microprocesorul in modul minim, intrarea $MN / \overline{MX}$ trebuie conectata la +5V. In acest mod de operare, CPU este optimizata sa functioneze in sisteme mici, mono-procesor, 8086 generand singur toate semnalele de control pe magistrala ($DT / \overline{R}$, $\overline{DEN}$, *ALE*, $M / \overline{IO}$, $\overline{RD}$, $\overline{WR}$ si $\overline{INTA}$) si, in plus, furnizand un mecanism pentru functia de acceptare cerere de magistrala compatibil cu un echipament de tip DMA (exemplu: controlerul DMA *INTEL*

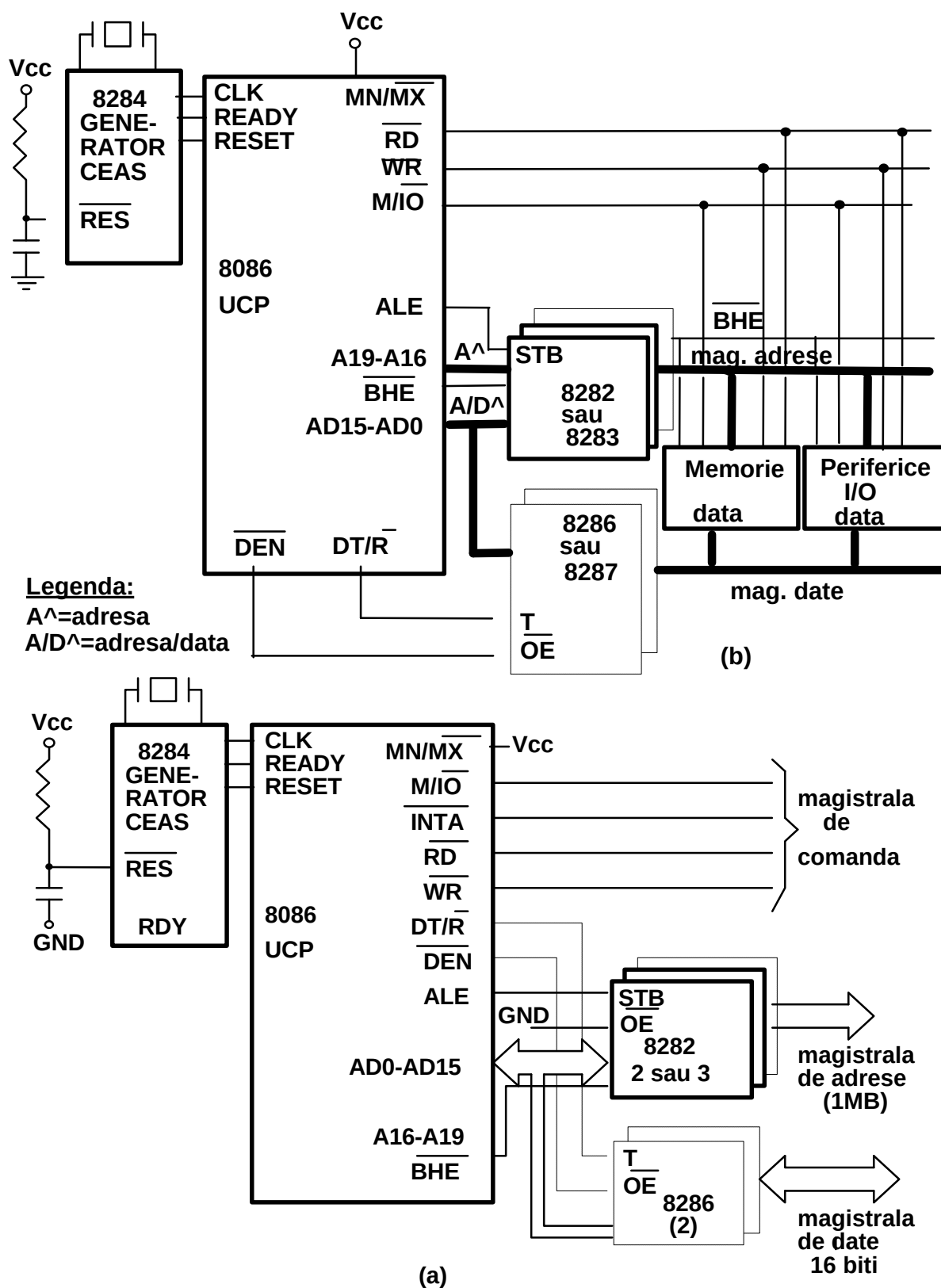


Fig. 4.22. Microprocesorul 8086 in modul minim.
 (a) Configurarea magistrelor; (b) conectarea resurselor sistemului

8257). In figura 4.22. se prezinta modul de configurare a magistralelor in modul minim (a), precum si un exemplu de sistem cu microprocesor 8086 functionand in modul minim (b).

Cand un echipament extern de tip DMA doreste sa efectueze un transfer direct intre un port de I/E si memorie, el semnalizeaza aceasta prin activarea liniei *HOLD* catre microprocesor. Ca raspuns la solicitarea de cedare a magistralei, 8086 isi activeaza iesirea *HLDA*, prin care semnalizeaza acceptarea cererii si cedarea magistralei simultan cu trecera tuturor iesirilor pe magistrala multiplexata si pe magistrala de control in stare de inalta impedanta. Cum o cerere de magistrala este un eveniment asincron, CPU esantioneaza intrarea *HOLD* pe fiecare tranzitie pozitiva a ceasului si, asa cum se arata in figura 4.23, isi activeaza iesirea *HLDA* fie la sfarsitul ciclului curent de magistrala, in cazul in care era vreunul in curs de desfasurare, fie la sfarsitul urmatoarei perioade T_1 . Microprocesorul se mentine in starea de *hold* pina cand "solicitantul de magistrala" dezactiveaza linia *HOLD*, moment in care CPU redobandeste controlul propriu asupra magistralelor de sistem. Oricum, si pe durata acestei stari microprocesorul continua sa execute instructiuni pana in momentul in care este necesar un ciclu de magistrala.

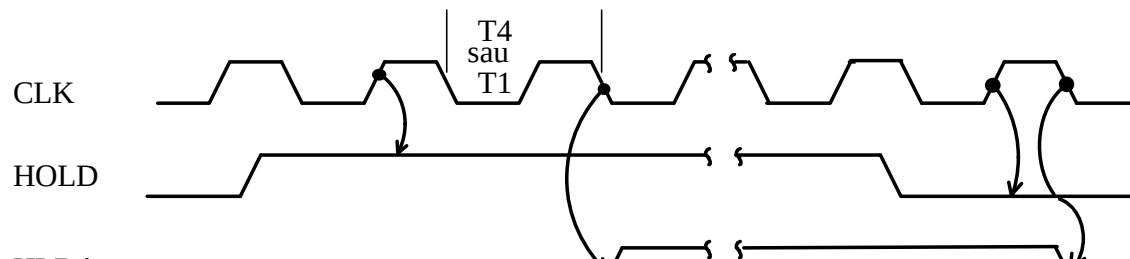


Fig. 4.23. Semnalele *HOLD* si *HOLDA*

MODUL MAXIM

Daca pinul $MN / \overline{MX}$ este conectat la masa, 8086 va opera in modul maxim in care se extinde arhitectura sistemului, pentru a suporta configuratii multi-procesor pe magistralele de sistem, precum si coprocesoare pe magistrala locala.

S2, S1, S0

Microprocesorul codifica semnalele de control pe trei linii de stare, S2-S0, si utilizeaza restul celor 5 linii ramase libere din cele care isi redefinesc semnificatia pentru a-si coordona activitatea pe magistrala cu alte microprocesoare din sistem.

Cele trei linii de stare sunt utilizate de un alt circuit din familia lui 8086 — **controlerul de magistrala INTEL 8288** — pentru a furniza semnalele de control si comanda pe magistralele sistemului.

In tabelul 4.5 este prezentata decodificarea liniilor $\overline{S2}, \overline{S1}, \overline{S0}$ si comanda generata de 8288 in fiecare stare, iar in figura 4.24, modul de configurare a magistralelor in modul maxim (a) si un exemplu de sistem cu microprocesor 8086 functionand in modul maxim (b).

Tabelul 4.5.

S2	S1	S0	CICLU MICROPROCESOR	IEsIRE 8288
0	0	0	Acceptare intrerupere	INTA
0	0	1	Citire port I/E	IORC
0	1	0	Scriere port I/E	IOWC, AIOWC
0	1	1	Halt	-
1	0	0	Extragere instructiune	MRDC
1	0	1	Citire memorie	MRDC
1	1	0	Scriere memorie	MWTC, AMWC
1	1	1	Pasiv	-

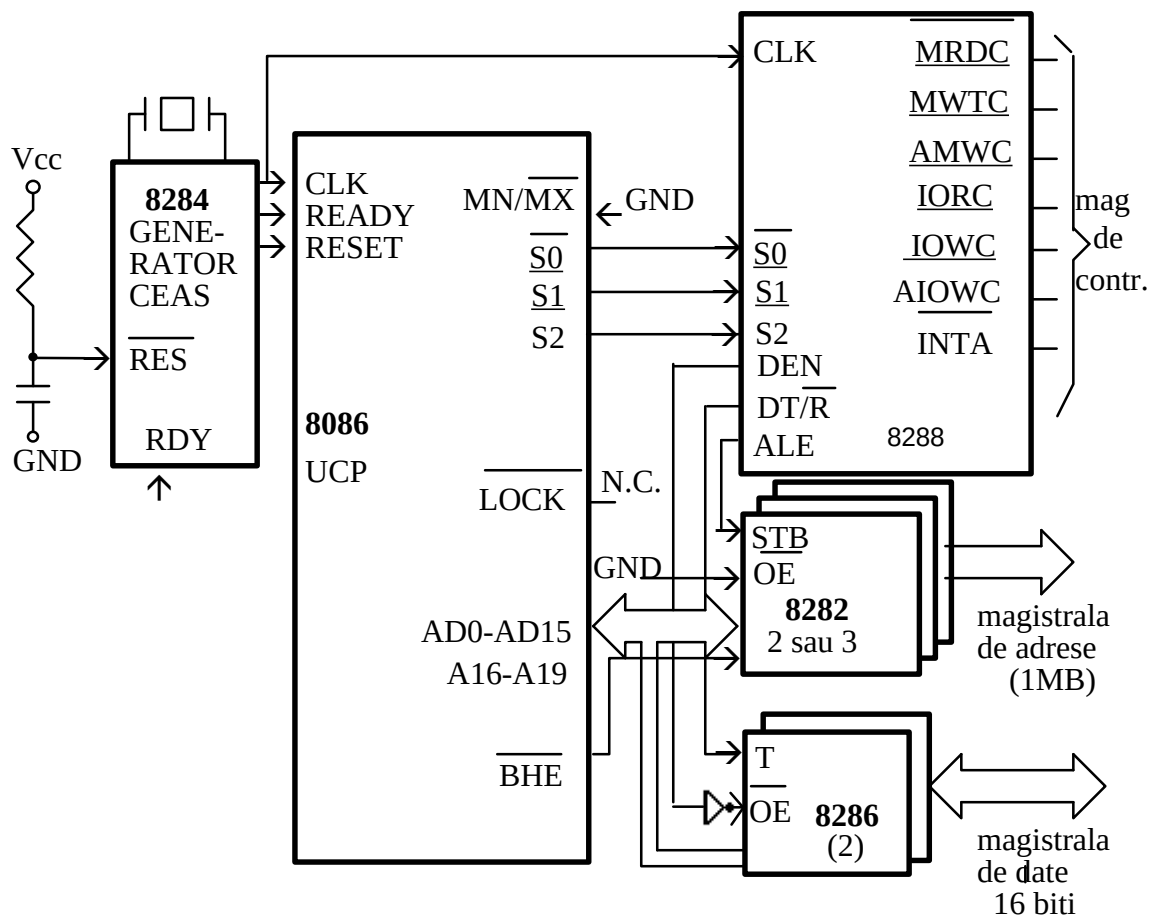


Fig. 4.24. Microprocesorul 8086 in modul maxim
(a) Configurarea magistralelor;

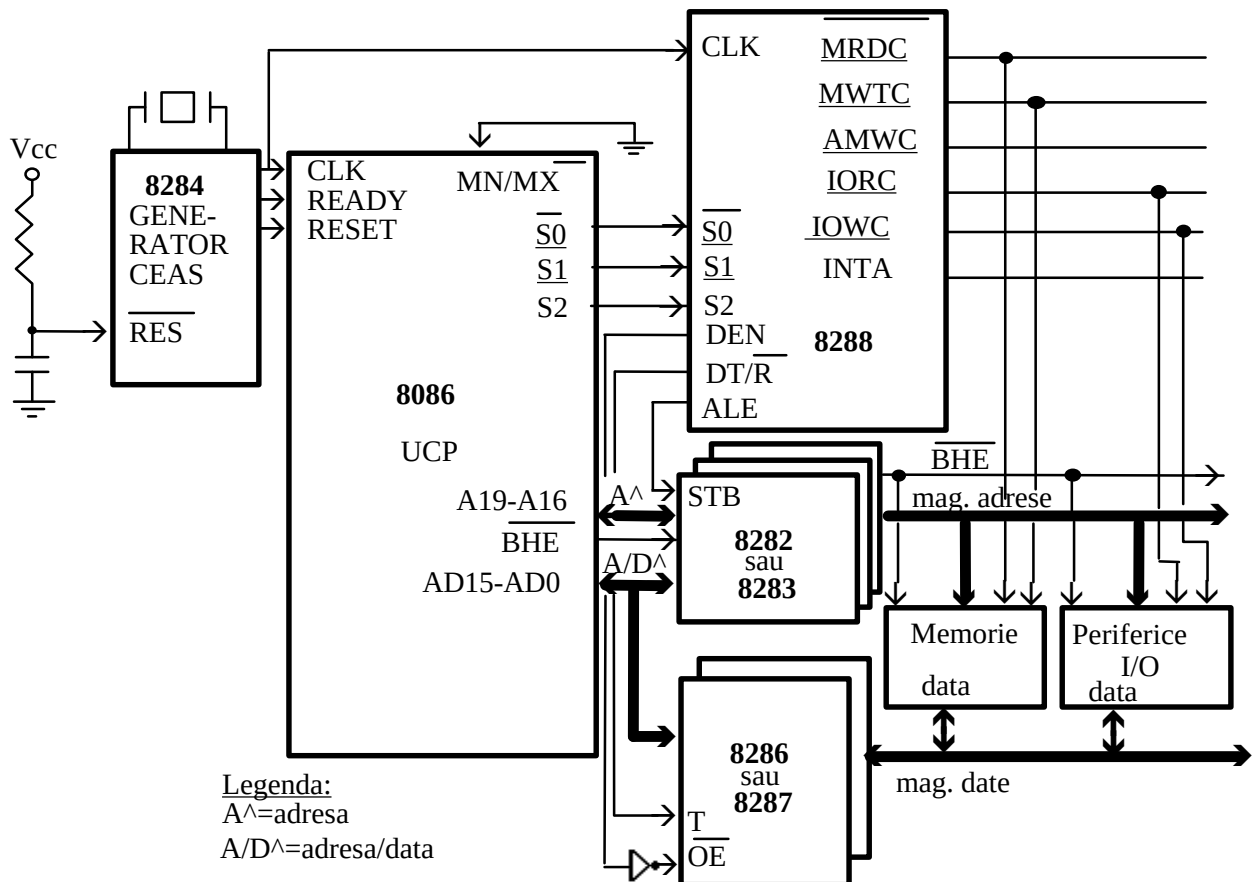
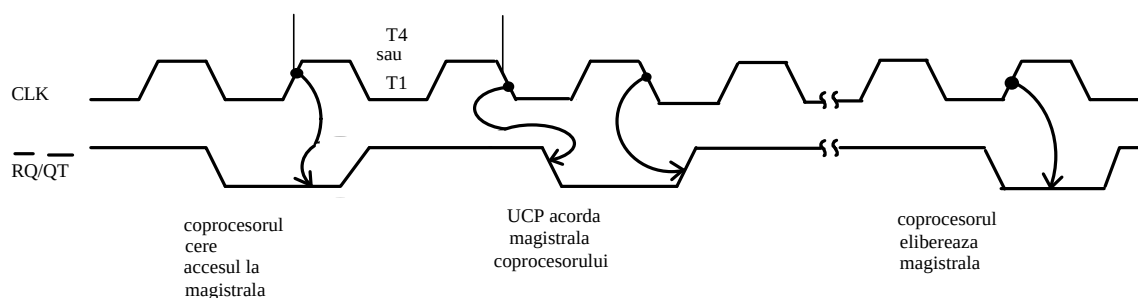


Fig. 4.24. Microprocesorul 8086 in modul maxim

 $\overline{RQ/GT1}$, $\overline{RQ/GT0}$

Liniile de cerere / acordare de magistrala (*request/grant*) implementeaza functia de acces la magistrala locala in cazul microprocesorului 8086 functionand in modul maxim in sisteme multi-procesor — inlocuind mecanismul *HOLD/HOLDA* existent in modul minim. Functia de cerere/acordare de magistrala necesita insa aici o singura linie, spre deosebire de cele doua din cazul precedent.

Asa cum se arata in figura 4.25, operatia se desfasoara intr-o secventa ce contine trei faze. Ea este initiata de un alt procesor din sistem — "solicitantul de magistrala", de obicei **coprocessorul aritmetic 8087** — ce furnizeaza un puls pe una din liniile *request/grant*, cerand astfel acces la magistrala (faza de cerere). Drept raspuns, CPU da la randul lui un puls (pe aceeaasi linie) la sfarsitul fie al ciclului curent de magistrala, daca asa ceva are loc in momentul respectiv, fie al starii T_1 in desfasurare. Prin aceasta, indica solicitantului ca si-a trecut *driver*-ele de magistrala in stare de inalta impedanta si ca se va deconecta logic de la controlerul de magistrala pe urmatoarea perioada de ceas (faza de acordare), intrand intr-o stare *hold*. 8086 continua sa execute intern instructiuni, fara sa se opreasca decat daca trebuie sa execute un ciclu de magistrala sau daca nu mai exista instructiuni in coada de asteptare. In a treia faza,

Fig. 4.25. Secvența temporală pe linia *request/grant*

cea de cedare, procesorul solicitant furnizează din nou un puls pe linia *request/grant*, semnaland

CPU ca este gata să elibereze magistrala. Pe următorul semnal de ceas (de remarcat, din figura 4.25, ca protocolul se desfășoară sincron cu același semnal de ceas *CLK*), 8086 reia controlul magistralei.

O ultimă mențiune este aceea că linia $\overline{RQ} / \overline{GT0}$ are prioritate mai mare decât linia $\overline{RQ} / \overline{GT1}$ care, la rândul ei — ca și *HOLD*, de altfel, în modul minim — este prioritară față de o cerere de întrerupere.

LOCK

Iesirea $\overline{LOCK}$ poate fi utilizată în sisteme multi-microprocesor în care accesul pe magistralele sistemului este controlat de un **arbitru de magistrală INTEL 8289**. Ea garantează accesul exclusiv pe magistrala comună, pe durata execuției unei instrucțiuni. Iesirea este controlată software prin precedarea instrucțiunii ce necesită acces exclusiv de instrucțiunea pe un octet — **LOCK**. Când decodifică acest prefix, unitatea de execuție informează unitatea de interfață cu magistrala să activeze iesirea $\overline{LOCK}$ începând cu următorul ciclu de ceas. Semnalul rămâne activ încă un ciclu de ceas după execuția instrucțiunii ce a accesat o resursă comună a sistemului multi-microprocesor (de exemplu, un *buffer*, un *pointer* sau un *semăfor*). Dacă accesul la resursa respectivă nu este controlat, un microprocesor ar putea citi o valoare eronată din resursa comună, în timp ce altul o actualizează.

QS1, QS0

QS1 și *QS0* sunt două linii de stare (*queue status*) ce permit monitorizarea de către coprocesorul aritmetic a stării în care se afla coada de instrucțiuni din BIU. Ele specifică ce activitate s-a executat, relativ la instrucțiunile din coada pe durata ciclului anterior de ceas (00 — nici un octet nu a fost extras din coadă; 01 — s-a extras primul octet al unei instrucțiuni; 10 — coada a fost reinitializată ca urmare a execuției unei instrucțiuni de transfer; 11 — s-a extras al doilea octet al unei instrucțiuni).

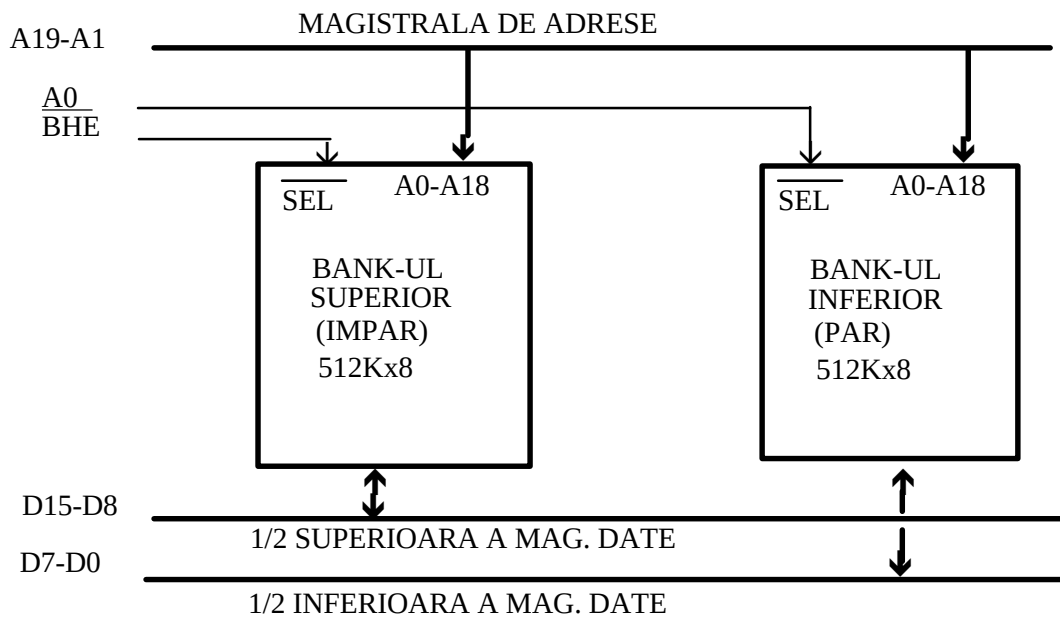
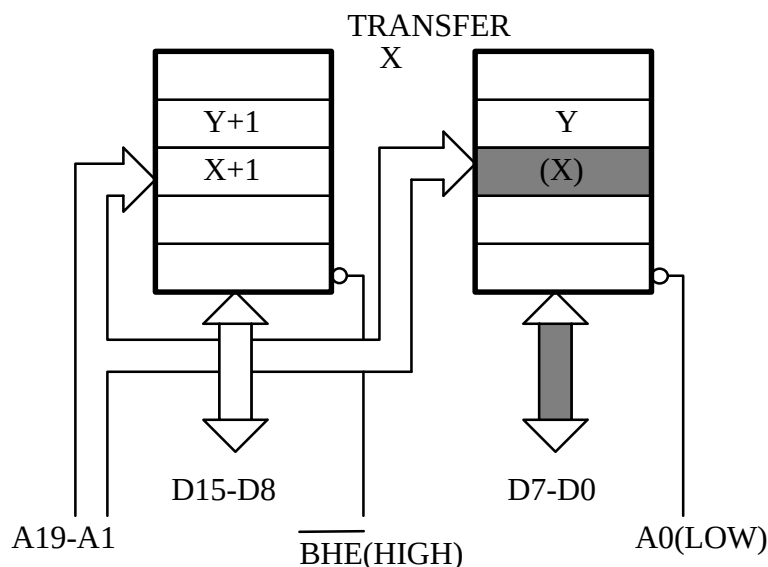
4.4.5 Selecția memoriei

Spatiul de memorie de un megabyte este divizat fizic în două zone (numite *bank*) a câte 512 Ko fiecare. Una din ele (*lower bank*) este asociată părții inferioare a magistralei de date (bitii D7-D0), iar cealaltă (*upper bank*) este conectată pe bitii cei mai semnificativi ai magistralei de date (D15-D8).

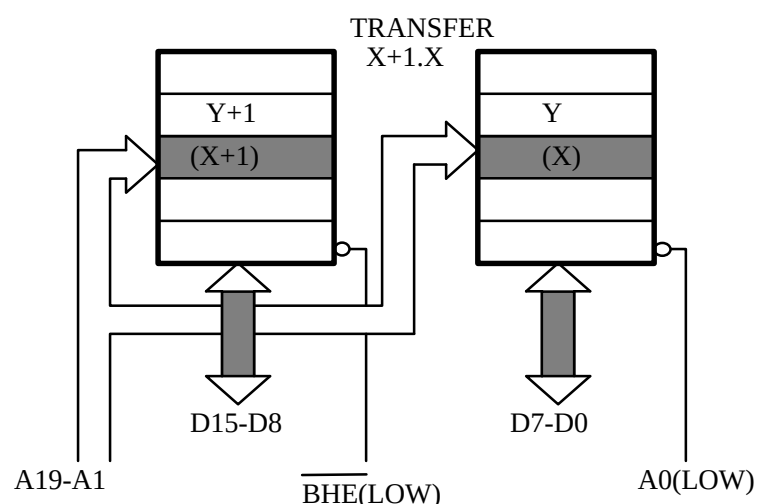
Tabelul 4.6. Mecanismul de transfer cu memoria

$\overline{BHE}$	A0	Octet transferat
0	0	Ambii octeti
0	1	Octetul high la / de la adresa impara
1	0	Octetul low la / de la adresa para
1	1	Nici unul

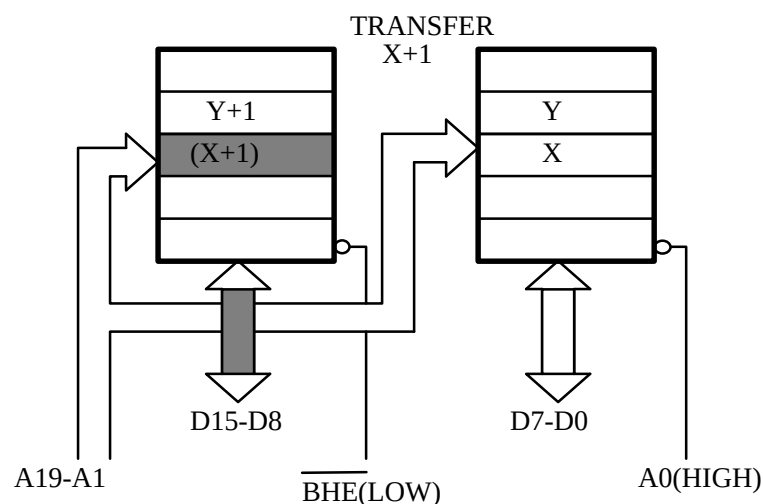
Liniile de adresa A19-A1 se utilizeaza pentru a adresa o locatie de un octet in mod simultan, atat in *lower bank* cat si in *upper bank*, in timp ce linia A0 nu se foloseste pentru adresarea locatiei in cadrul unui *bank*, ci pentru selectia *bank*-ului. *Bank*-ul inferior, care contine octeti aflatii la adrese pare, este selectat cind $A0=0$. *Bank*-ul superior, continand octeti situati la adrese impare, este selectat de semnalul $\overline{BHE}$ (semnifica transferul unui octet pe liniile D15-D8), in conditia $\overline{BHE} = 0$. Acest mecanism de selectie este ilustrat in tabelul 4.6 si in figura 4.26.

Fig. 4.26. Selectia *bank*-urilor de memorie

Cand se acceseaza un octet la o adresa para, acesta este transferat pe liniile inferioare ale magistralei, D7-D0. In aceasta situatie, nivelul inactiv al liniei de adresa A0 ($A0=0$) permite selectia locatiei aflata in *bank*-ul inferior; in acelasi timp, nivelul inactiv al semnalului $\overline{BHE}$ ($\overline{BHE} = 1$) impiedica selectia locatiei aflata in *bank*-ul superior.



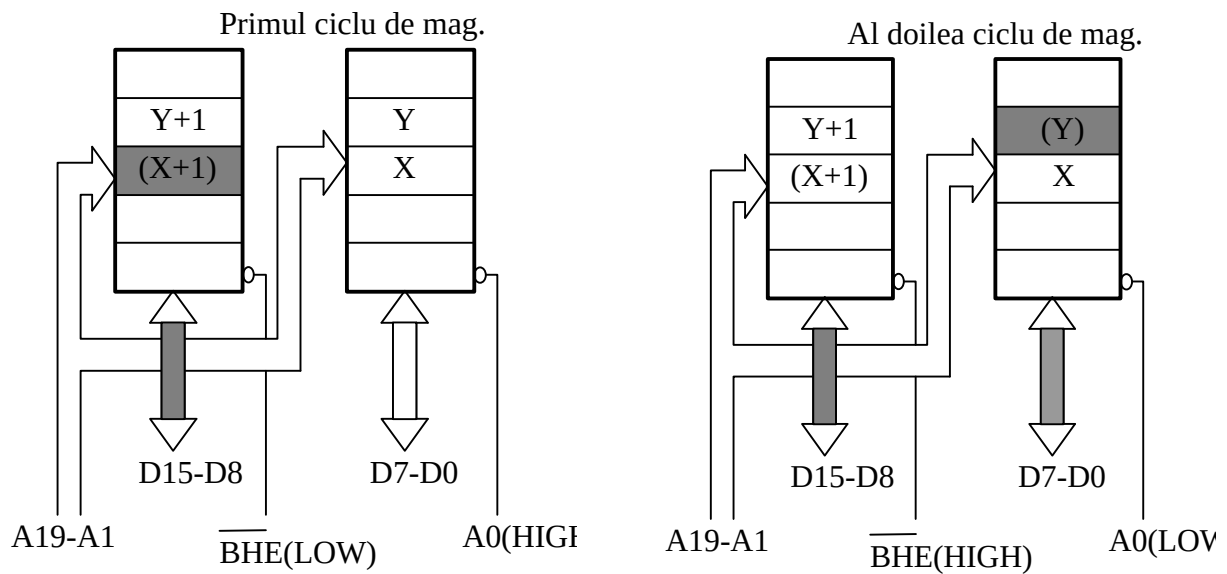
În mod similar, când se accesează un octet la o adresă impară, acesta este transferat pe liniile superioare ale magistralei, D15-D8. Acum nivelul activ al lui $\overline{BHE}$ ($\overline{BHE} = 0$) permite selecția locației aflate în *bank*-ul superior, în timp ce nivelul activ al liniei de adresă **A0** (**A0=1**) împiedică selecția *bank*-ului inferior.



Așa cum se arată în tabelul 4.6, **8086** poate accesa o locație din *lower bank* simultan cu una din *upper bank* pentru a efectua transferul unei date de tip *word*. Când octetul *low* al cuvântului ce trebuie transferat se află la o adresă pară (deci se află în *lower bank*), cuvântul este **aliniat** și poate fi transferat într-un singur ciclu de magistrală. Liniile A19-A1 adresează locația

corespunzătoare din ambele *bank*-uri, *bank*-uri care sunt amândouă selectate simultan, cel inferior prin **A0=0** iar cel superior prin $\overline{BHE} = 0$.

Când octetul mai puțin semnificativ al cuvântului se află la o adresă impară, cuvântul este **nealiniat** și se va transfera în două cicluri de magistrală. În prima fază, octetul *low* este transferat pe biții B15-B8, cu semnalele de selecție **A0=1** și $\overline{BHE} = 0$. Adresa de memorie este incrementată în a doua fază, ceea ce face **A0=0** și octetul *high* aflat în *lower bank* va fi transferat pe liniile D7-D0. Aceasta secvență este inițiată automat de **8086** ori de câte ori transfera un cuvânt la o adresă impară, dar cu excepția celor 4 perioade de ceas suplimentare, necesare execuției celui de-al doilea ciclu de magistrală, întreaga operație este transparentă pentru program.



4.4.6 Selectia si interfatarea porturilor de intrare/iesire

Nu este obligatoriu ca un port de I/E utilizat intr-un sistem cu microprocesor 8086 sa dispuna de 16 linii pentru conectare pe magistrala de date, ci se pot utiliza si porturi pe 8 biti.

Porturile pe 8 biti se pot lega fie pe liniile D15-D8, fie pe D7-D0. Pentru a nu aparea o supra-incarcare a *driver*-elor de magistrala, este recomandabil ca circuitele de I/E pe 8 biti sa fie distribuite in mod egal pe ambele parti ale magistralei de date.

Daca un port este conectat pe liniile D7-D0, toate adresele de I/E care ii sunt asiguate trebuie sa fie pare ($A0=0$). In mod similar, toate adresele asiguate unui port conectat pe liniile D15-D8 trebuie sa fie impare ($A0=1$). Aceasta permite transferarea unui octet pe liniile superioare sau inferioare ale magistralei de date, dupa cum adresa portului selectat este para, respectiv impara. Rezulta de aici ca $A0$ nu poate fi utilizata ca o linie de adresa de port pentru a selecta eventuale registre sau canale ale acestuia, ci impreuna cu $\overline{\text{BHE}}$ va determina obtinerea selectiei de circuit ($\overline{\text{CS}}$). Cateva tehnici in acest sens sunt prezentate in figura 4.27.

Prima tehnica (a) utilizeaza 2 decodificatoare separate de tip 8205 pentru a genera semnalele de selectie de circuit pentru porturi pe 8 biti situate la adrese pare sau impare, precum si pentru porturi pe 16 biti situate la adrese pare (un port pe 16 biti poate fi constituit si din 2 porturi pe 8 biti situate la adrese consecutive).

Schema (b) permite selectia de circuite de I/E doar pentru transferuri de octet la porturi pe 8 biti, indiferent la ce adresa se afla.

Schema (c) implementeaza aceeasi functie de selectie ca si (a), dar utilizeaza un singur decodificator.

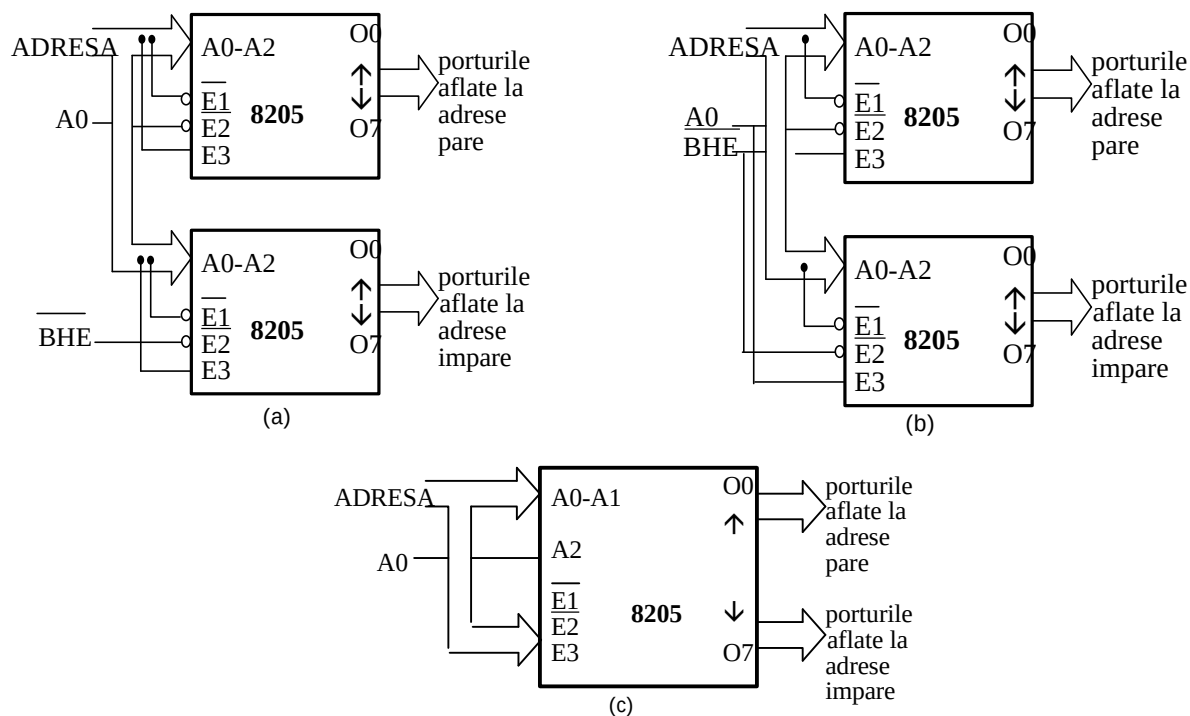


Fig. 4.27. Tehnici pentru selectia porturilor de I/E

O ultima tehnica pentru interfatarea porturilor pe 8 biti situate la adrese succesive este prezentata in figura 4.28. Magistrala de date de 16 biti este multiplexata, cu ajutorul a 2 circuite tampon bidirectionale cu iesiri 3-state de tip 8286, intr-o magistrala de 8 biti utilizata de circuitele periferice pe 8 biti. Aceasta permite efectuarea de transferuri DMA orientate pe octet, sau transferuri la porturi de I/E pe 8 biti suprapuse peste memorie (*memory-mapped I/O*).

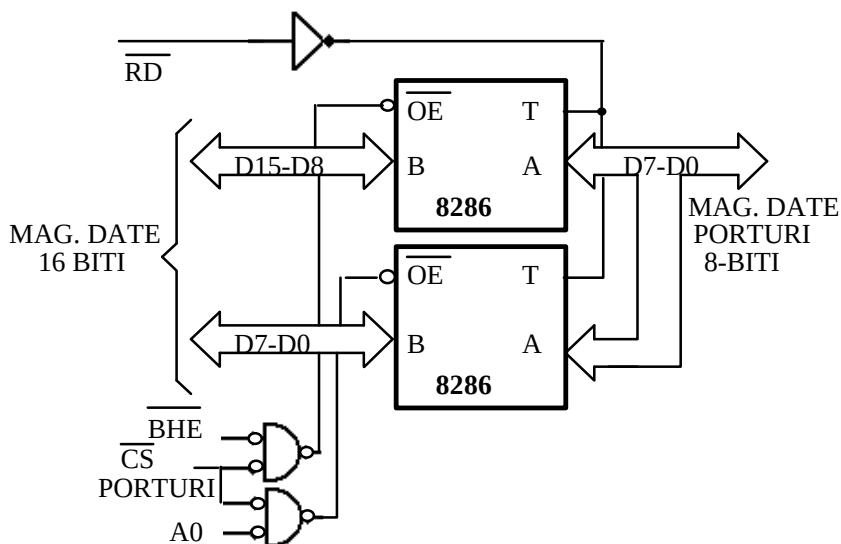


Fig. 4.28. Conversie de magistrala 16-8 biti

În ceea ce privește porturile pe 16 biți (sau, mai general, porturile pe mai mult de 8 biți: 10, 12 sau 16), este evident că, din motive de eficiență a utilizării magistralei și de simplitate a selecției echipamentelor, ele trebuie situate la adrese pare. Pentru a garanta că portul este utilizat doar pentru transferuri de date de tip *word*, **A0** și **BHE** trebuie să conditioneze generarea de semnale de selecție de circuit, o posibilă implementare a obținerii semnalelor de selecție fiind ilustrată în figura 4.29.

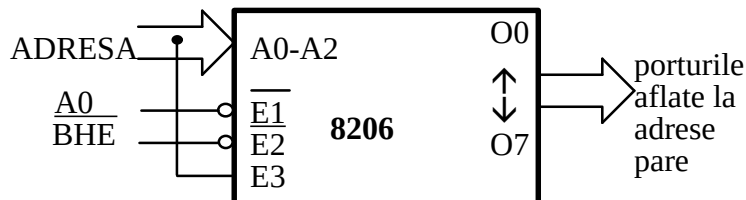


Fig. 4.29. Decodificarea porturilor de I/E pe 16 biți

4.5 Sistemul de întreruperi

Așa cum s-a arătat în capitolul 3, una din cele mai utilizate metode pentru a efectua operațiile de I/E se bazează pe utilizarea tehnicii întreruperilor.

Prin această metodă, execuția programului principal de către CPU este oprită în mod asincron de către un echipament de I/E care solicită, printr-o **cerere de întrerupere**, un tratament preferențial (tratament care să rezolve cauza ce a generat apariția **evenimentului semnificativ** ce a determinat cererea de întrerupere). Procesorul termină de executat instrucțiunea curentă și, prin intermediul unui **vector de întrerupere**, ajunge la **rutina de tratare a întreruperii** respective în care servește echipamentul ce a solicitat întreruperea. La sfârșitul acestei subrutine, procesorul revine în programul principal, reluându-l de la instrucțiunea următoare celei ce a fost executată înainte de acceptarea întreruperii. Prin acest mecanism, operațiile de intrare/ieșire sunt executate practic fără întârziere față de momentul în care apare necesitatea efectuării lor. Aceasta implică însă existența unui *hardware* suplimentar, care să permită implementarea unui astfel de mod de operare (circuitul **INTEL 8259 — Programmable Interrupt Controller** — este special proiectat în acest scop, fiind pe deplin compatibil cu sistemul de întreruperi al lui 8086).

În plus față de întreruperile generate de porturile de I/E, care aparțin clasei întreruperilor externe, 8086 poate fi solicitat și de întreruperi generate intern, ca urmare a execuției programului.

4.5.1 Sursele de generare a întreruperilor

Microprocesorul 8086 acceptă **întreruperi externe** (generate de un echipament extern) și **întreruperi interne** — generate intern în CPU fie prin execuția unor instrucțiuni specifice (întreruperi software), fie ca urmare a împlinirii unor condiții specifice la nivelul microprocesorului.

Fiecarei intreruperi ii este atribuit un **cod** numit **tipul intreruperii**, care permite identificarea acesteia de catre microprocesor. *8086* poate manipula pana la 256 tipuri diferite de intreruperi. Figura 4.30 prezinta posibilele surse de intrerupere intr-un sistem cu microprocesor *8086*, iar figura 4.31 ilustreaza secventa de raspuns a logicii de control al intreruperilor.

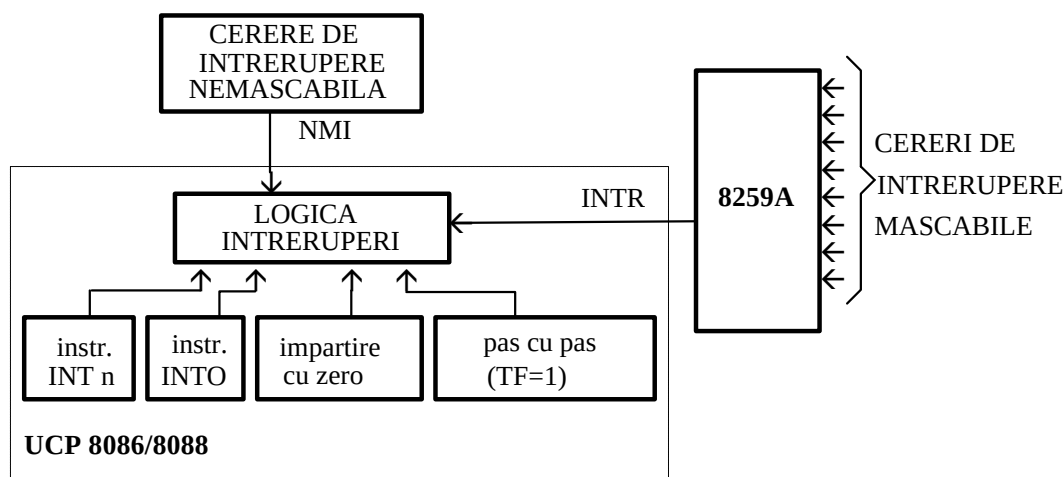


Fig. 4.30. Sursele de intrerupere

4.5.2 Vectorii de intrerupere

Legatura dintre tipul intreruperii si procedura care deserveste intreruperea respectiva este reprezentata de **tabela vectorilor de intrerupere**. Aceasta tabela ocupa primul ko de memorie incepand cu adresa zero si are pana la 256 de intrari, corespunzand la 256 de vectori de intrerupere—cate unul pentru fiecare tip de intrerupere ce poate fi definit intr-un sistem cu microprocesor *8086*.

Un **vector de intrerupere** al microprocesorului *8086* reprezinta un pointer (4 octeti) continand adresa rutinei de tratare a intreruperii asociate. Cuvantul cel mai semnificativ al pointerului contine adresa de baza de segment — si se va incarca in registrul CS — iar cuvantul mai putin semnificativ contine ofsetul fata de inceputul segmentului al subrutinei respective — ofset ce se va incarca in registrul IP — astfel incat urmatoarea instructiune ce se va extrage si executa va fi prima instructiune din cadrul subrutinei de intrerupere. Cum fiecare intrare in tabela vectorilor de intrerupere are o lungime de 4 octeti, CPU calculeaza locatia vectorului asociat unei intreruperi anume prin simpla inmultire cu 4 a numarului (cuprins intre 0 si 255) ce reprezinta **tipul** intreruperii respective (figura 4.32.).

Primii 5 vectori de intrerupere sunt **dedicati** intreruperilor generate prin software, precum si unei intreruperi externe, *NMI (0—divide error, 1—single-step, 2—NMI, 3—breakpoint, 4—overflow)*, urmasorii 27 (deci pina la locatia 07FH) sunt **rezervati** de firma *INTEL*, iar restul (vectorii 32 pina la 255) sunt la dispozitia utilizatorului.

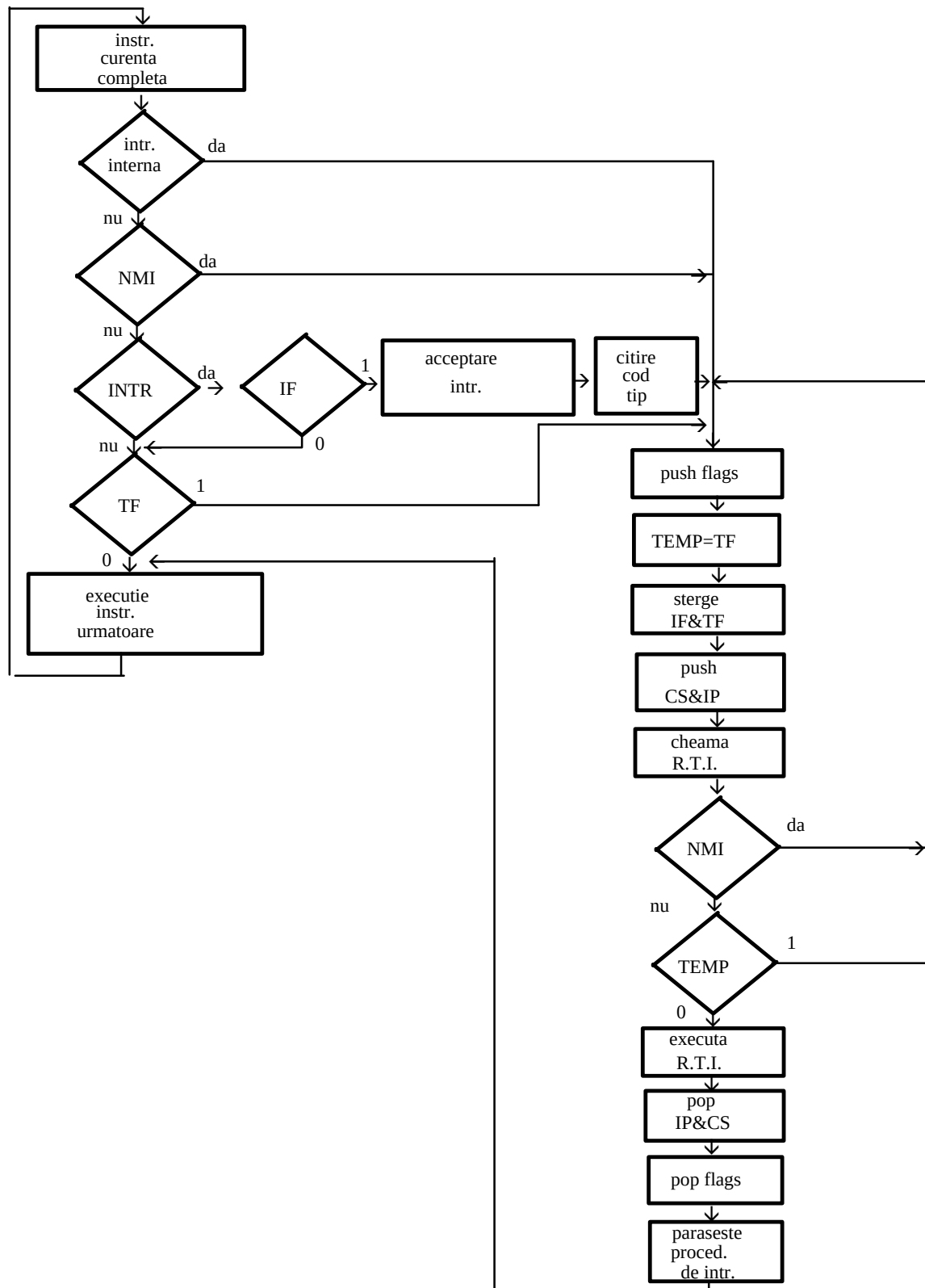


Fig. 4.31. Secventa de raspuns la intrerupere

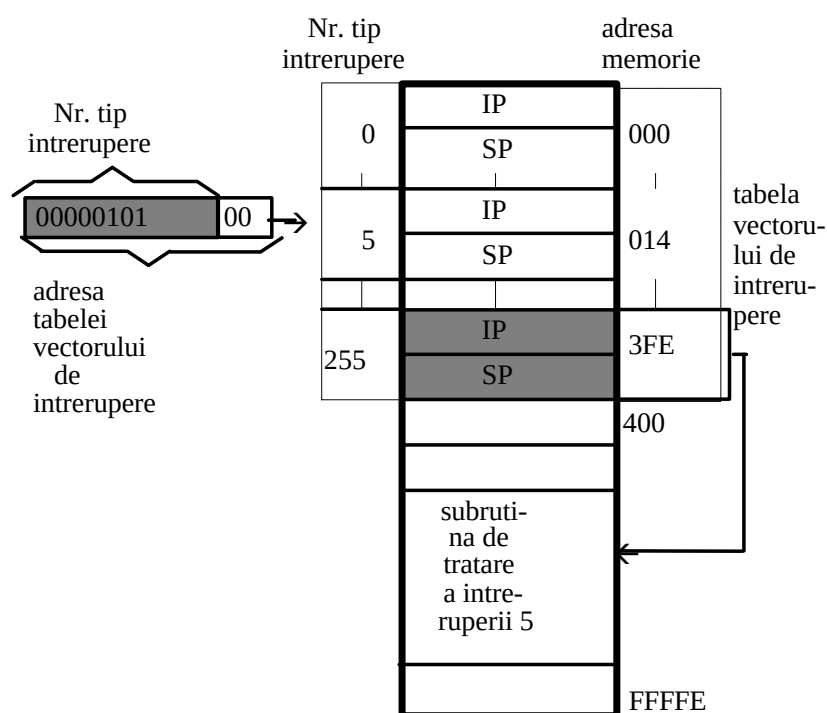


Fig. 4.32. Directionarea catre rutina de tratare a intreruperii prin intermediul tabelii vectorilor de intrerupere

4.5.3 Intreruperile externe

Microprocesorul 8086 pune la dispozitia surselor externe de intrerupere doua linii de intrare prin care se poate comunica CPU o cerere de intrerupere.

Una este linia *NMI* (cerere de **intrerupere nemascabila**), o intrerupere pe aceasta cale aparand ca rezultat al tranzitiei pozitive a semnalului *NMI*. Cum intrarea este asincrona, pentru a fi sigur ca este recunoscuta, *NMI* trebuie sa aiba o durata de minimum 2 perioade de ceas. Cererea de intrerupere nemascabila este utilizata, de obicei, pentru a semnala microprocesorului aparitia unui eveniment "catastrofal" ce semnifica existenta unui pericol major pentru buna functionare a sistemului. Exemple tipice sunt iminenta cadere a tensiunii de alimentare, aparitia unei erori de memorie sau a unei erori de paritate pe magistrala.

Deoarece intreruperii *NMI* ii este asociat tipul 2 de intrerupere, la acceptarea acesteia — la sfirsitul executiei instructiunii curente — controlul programului este transferat rutinei de tratare a intreruperii nemascabile a carei adresa este data de vectorul 2. Aceasta implica salvarea in stiva a continutului registrului indicatorilor de conditie (*SP* este decrementat cu 2) si apoi resetarea indicatorilor *IF* si *TF* (dezactivandu-se, deci, automat intreruperile mascabile si intreruperea pentru modul de operare pas-cu-pas). Registrele *CS* si *IP* sunt incarcate cu adresele continute de vectorul 2, urmand extragerea si executia primei instructiuni din rutina de tratare a intreruperii nemascabile.

A doua cale prin care sistemul extern de intreruperi poate emite o cerere de intrerupere este linia *INTR*. Aceasta este de obicei activata de **controlerul de intreruperi 8259A** ale carui sarcini, in ceea ce priveste declansarea unui proces de intrerupere, sunt urmatoarele:

- primește cererile de intrerupere de la echipamentele de I/E atasate la el;
- determina care dintre solicitanti (daca apar mai mult de unul, la un moment dat) are cea mai inalta prioritate;
- activeaza linia *INTR* catre microprocesor daca solicitantul selectat are un nivel de prioritate mai mare decit al celui care este servit in acel moment (daca exista vreunul in aceasta situatie).

Chiar si numai cele aratate mai sus sugereaza faptul ca *INTEL 8259A* este un circuit programabil, fiind controlat prin *software* de catre programul executat de *8086* in care controlerul de intreruperi este privit ca o interfata specializata (un set de porturi de I/E) — v. figura 4.33. In plus, asa cum se va vedea in continuare, circuitul are un rol important in specificarea vectorului asociat unei cereri de intrerupere acceptate de catre microprocesor.

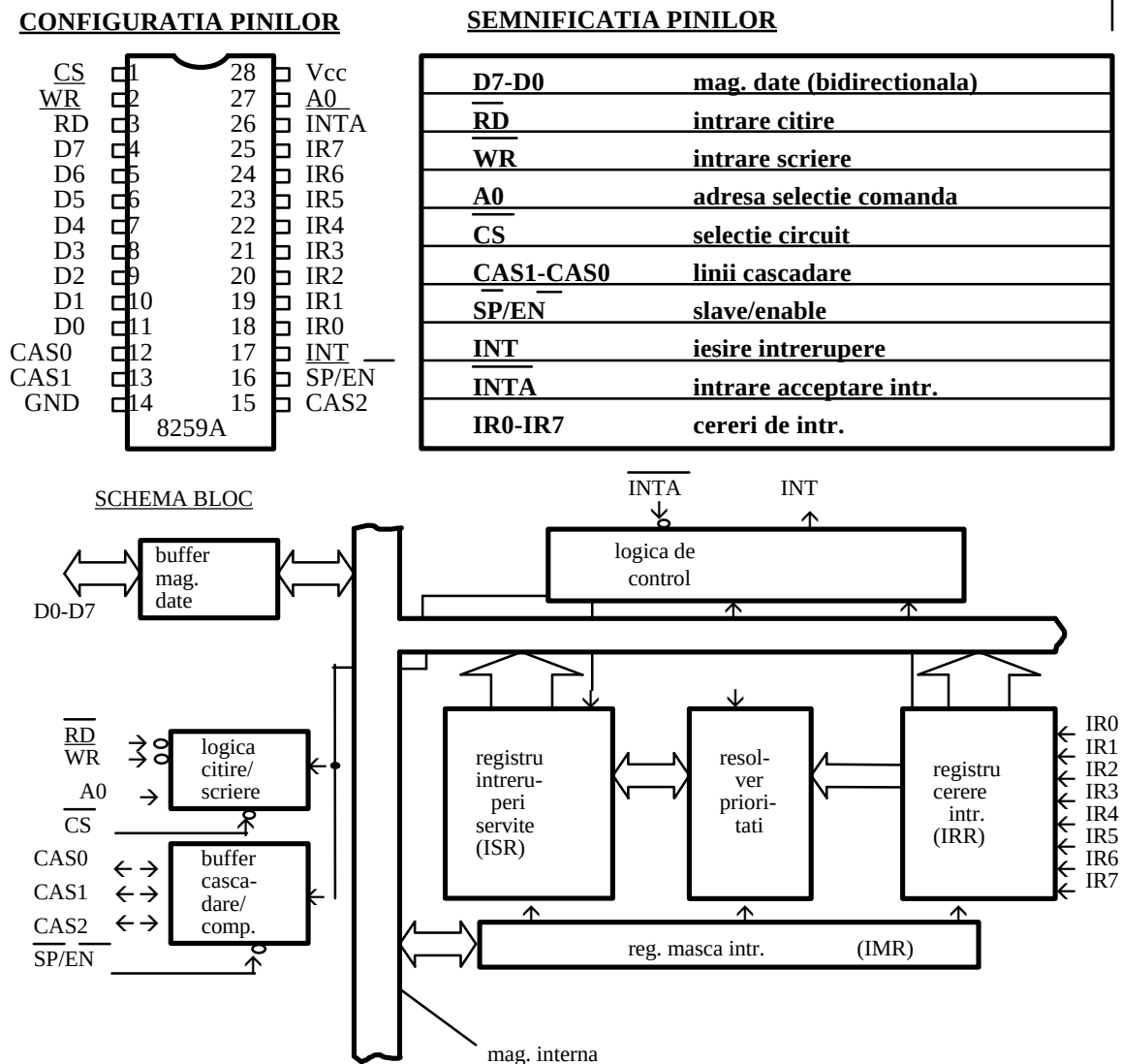


Fig. 4.33. Schema bloc si configuratia pinilor circuitului *INTEL 8259A*

Aparitia semnalului *INTR* cauzeaza actiuni diferite ale CPU in functie de starea indicatorului de conditie de activare a intreruperii mascabile (*IF*). Starea acestuia este controlata de instructiunile *STI* (*IF=1*) si, respectiv, *CLI* (*IF=0*). Nici o actiune nu are loc insa pana la terminarea executiei instructiunii curente. Apoi, daca *IF=0* (ceea ce inseamna ca intreruperile ce apar pe linia *INTR* sunt mascate, dezactivate) CPU ignora cererea de intrerupere si continua cu executia urmatoarei instructiuni. Este de notat faptul ca semnalul *INTR* nu este memorat in vreun circuit *latch* in microprocesor, astfel incat el trebuie mentinut activ pana se primeste un raspuns sau cererea este retrasa. Din punct de vedere al specificatiei de semnal, intrarea *INTR* este activa pe nivel si triggerata pe frontul pozitiv al ceasului *CLK*. Ea trebuie sa fie deci activa pe durata perioadei de ceas ce precede sfarsitul executiei instructiunii curente.

Daca intreruperile sunt activate (deci daca *IF=1*), atunci CPU recunoaste intreruperea (o accepta) si urmeaza sa o proceseze. In afara instructiunilor *STI* si *CLI*, intreruperile ce sosesc la controlerul de intreruperi pot fi selectiv mascate (unele inhibitate, altele activate) prin cuvinte de comanda trimise catre 8259A — cuvinte ce programeaza starea bitilor unui registru de mascare a cererilor de intrerupere ce exista la nivelul acestuia.

Asa cum se arata in figura 4.34, recunoasterea unei intreruperi mascabile implica executia de catre microprocesor a doua **cicluri de magistrala de acceptare a intreruperii** — *INTA cycles*.

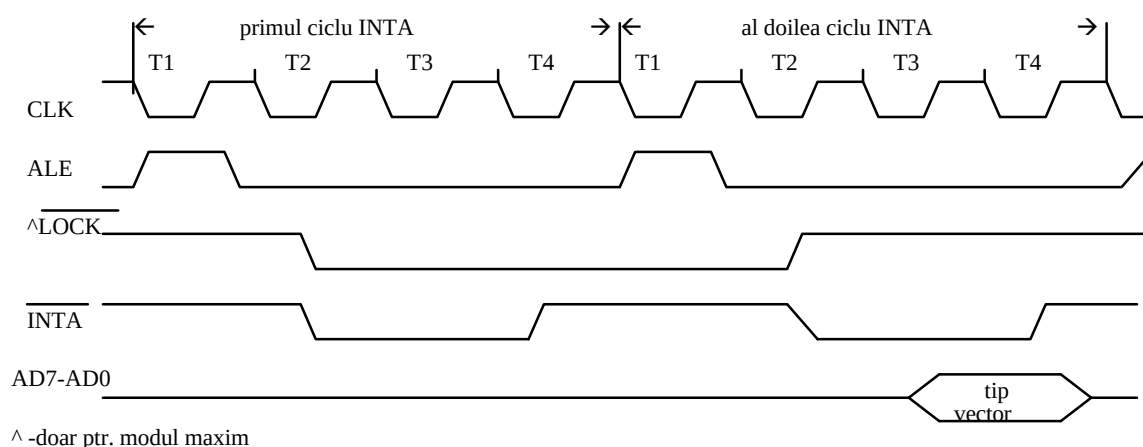


Fig. 4.34. Secventa de recunoastere a intreruperii *INTR*

Pe durata primului ciclu, CPU isi trece in stare de inalta impedanta *driver*-ele de magistrala si furnizeaza semnalul de control $\overline{\text{INTA}}$ (*INTerrupt Acknowledge*) din starea T2 pana in starea T4. Daca este in modul minim, microprocesorul va refuza recunoasterea unei cereri de magistrala (*HOLD*) pina la terminarea completa a secventei de acceptare a intreruperii. Daca este in modul maxim, 8086 isi activeaza iesirea $\overline{\text{LOCK}}$ incepand cu starea T2 a primului ciclu si pana in starea T2 a celui de-al doilea ciclu, pentru a semnala unui eventual arbitru de magistrala (de tip *INTEL 8289*) interzicerea accesului unui alt procesor pe magistrala.

Rolul acestui prim ciclu $\overline{\text{INTA}}$ este de a avertiza controlerul de intreruperi ca cererea transmisa pe linia *INTR* este onorata de microprocesor.

8259A este gata acum sa "participe" la transferul executiei programului catre rutina de tratare a intreruperii in cauza. Aceasta se realizeaza pe parcursul celui de-al doilea ciclu $\overline{INTA}$ initiat de catre microprocesor, in care 8259A plaseaza pe liniile cele mai putin semnificative ale magistralei de date (AD7-AD0) un octet ce contine tipul intreruperii asociate liniei **Interrupt Request** (vezi figura 4.33) activata de echipamentul ce a solicitat intreruperea. In acest mod se realizeaza identificarea sursei externe ce a generat intreruperea.

Tabelul 4.7. Octetul vectorului de intrerupere

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

In tabelul 4.7 este ilustrat modul de constituire a octetului respectiv. Bitii D7-D3 sunt la dispozitia utilizatorului si sunt incarcati printr-un cuvint de comanda la programarea circuitului. Bitii D2-D0 sunt automat inserati de 8259A, pentru a specifica pe care din cele 8 intrari $IR7-IR0$ a primit o cerere de intrerupere in urma careia a generat

semnalul $INTR$ catre microprocesor, cauzand astfel cele doua cicluri $\overline{INTA}$. Octetul in discutie (reprezentind chiar tipul intreruperii) este citit de CPU intocmai ca intr-un ciclu $READ$ (de altfel, in ambele cicluri $\overline{INTA}$ este acompaniat de semnalele $DT / \overline{R}$ si DEN), valoarea lui multiplicata cu 4 indicand adresa din tabela vectorilor de intrerupere unde se gaseste (prin grija programatorului !) adresa rutinei de tratare a intreruperii determinata de tipul respectiv.

Saltul efectiv la rutina de serviciu se face din acest moment la fel ca si in cazul intreruperii nemascabile (procedeul este valabil si pentru intreruperile interne): se salveaza in stiva indicatorii de conditie, se reseteaza IF si TF , se salveaza CS si IP si se incarca noile valori ale registrelor CS si IP din tabela vectorilor de intrerupere.

Intreruperea $INTR$ are prioritate mai mica decit intreruperea NMI .

4.5.4 Intreruperile interne

Instructiune **INT *n*** genereaza o intrerupere de tip ***n*** prin insasi executia sa. Vectorul intreruperii respective nu mai trebuie furnizat de nimeni, intrucat se afla codificat direct in instructiune.

Daca indicatorul de conditie OF este setat, instructiunea **INTO** (**INTerrupt on Overflow**) genereaza o intrerupere de tip **4**, a carei rutina de serviciu trebuie sa trateze situatia aparitiei unei depasiri.

CPU insasi genereaza o intrerupere al carei tip este **0** (*divide error*) imediat dupa executia instructiunilor **DIV** (**DI**VIde) sau **IDIV** (**I**nteger **DI**VIde), daca destinatia specificata in instructiune pentru a memora catul impartirii nu are dimensiunea suficienta pentru acest scop.

Daca indicatorul de conditie *TF* este setat, microprocesorul genereaza automat o intrerupere de tip **1** dupa **fiecare** instructiune, intrind astfel in modul de operare pas-cu-pas (*single-step*). Intreruperea de tip **1** constituie astfel o "unealta" puternica pentru depanarea programelor intrucat rutina de tratare se poate constitui ca o "fereasta in sistem" prin care executia programului sa poata fi urmarita instructiune cu instructiune. O rutina de tratare a intreruperii *single-step* poate afisa continutul registrelor microprocesorului, variabile semnificative etc., urmarindu-le evolutia pe parcursul executiei programului pentru a determina punctul in care apare o functionare nedorita.

Ultima intrerupere dedicata este cea de tip **3** — intreruperea de *breakpoint*. Un *breakpoint* reprezinta un punct in program unde executia este oprita pentru a se efectua anumite procesari speciale — de exemplu, trecerea in modul de operare pas-cu-pas pentru a depana o zona de program suspectata de erori. Avind doar lungimea unui octet (cod CCH), instructiunea *INT 3* poate fi usor "introdusa" intr-un program in curs de depanare.

Toate instructiunile interne (unele exceptii prezinta intreruperea pas-cu-pas) au urmatoarele caracteristici:

1. Codul tipului intreruperii este fie predefinit, fie continut in instructiune.
2. **Nu** are loc nici un ciclu de magistrala de recunoastere a intreruperii.
3. O intrerupere interna nu poate fi dezactivata (cu exceptia intreruperii pas-cu-pas ce este inhibata de resetarea indicatorului *TF*).
4. Orice intrerupere interna (cu exceptia intreruperii pas-cu-pas) are prioritate mai mare decit orice intrerupere externa. Ordinea de prioritate este:
 - I. intrerupere interna (cu exceptia pas-cu-pas)
 - II. *NMI*
 - III. *INTR*
 - IV. intreruperea pas-cu-pas

4.5.5 Subrutina de tratare a intreruperii

Cand se intra intr-o rutina de tratare a intreruperii, am vazut ca se salveaza in stiva in mod automat indicatorii de conditie, registrele *CS* si *IP*, iar *TF* si *IF* sunt resetati. Procedura respectiva poate reactiva intreruperile externe cu instructiunea *STI*, permitand sa fie ea insasi intrerupta de o cerere pe linia *INTR*, si desigur ca o rutina de tratare a intreruperii poate oricand fi suspendata de o cerere de intrerupere nemascabila. Acelasi efect il are si aparitia unei intreruperi interne. Trebuie evitata posibilitatea ca o intrerupere de un anumit tip sa-si intrerupa propria rutina de serviciu a intreruperii (de exemplu, o incercare de impartire la zero in rutina de tratare a intreruperii de tip **0** ar avea ca efect reintrarea continua in procedura respectiva).

Subrutina de tratare a unei intreruperi trebuie sa salveze toti registrii pe care ii utilizeaza inainte de a-i initializa si sa-i restaureze inainte de terminarea rutinei. Ca urmare, in cazul procesarii unor intreruperi simultane, stiva trebuie sa aiba la dispozitie un spatiu suficient pentru salvarile succesive ce pot aparea. O alta problema ce trebuie avuta in vedere este faptul

ca dezactivarea intreruperilor externe intr-o rutina de tratare a unei intreruperi poate duce la pierderea acestora, daca rutina are un cod prea mare.

Toate subrutinele de intrerupere trebuie sa se termine cu instructiunea *IRET* (***Interrupt RETurn***), instructiune a carei executie se bazeaza pe ipoteza ca stiva este in aceeaasi conditie in care a fost la intrarea in procedura. *IRET* extrage din stiva trei cuvinte succesive pe care le incarca in *IP*, *CS* si registrul indicatorilor de conditie, determinand astfel reluarea executiei programului cu instructiunea ce urma in mod logic, daca nu s-ar fi efectuat saltul la intrerupere.

Ceea ce se proceseaza efectiv in cadrul unei rutine de tratare a intreruperii depinde de aplicatia respectiva. De exemplu, in cazul unei proceduri care serveste o cerere de intrerupere externa, prima actiune ce trebuie executata (daca acest lucru nu se intampla automat) este trimiterea unei comenzi catre echipamentul ce este servit, prin care sa se determine retragerea cererii de intrerupere. Urmatoarea actiune uzuala este de a se citi starea echipamentului pentru a se identifica motivul pentru care s-a solicitat o intrerupere. In fine, in functie de cauza respectiva, se comanda executia unor operatii corespunzatoare.

5. IMPLEMENTAREA INTERFETELOR DE INTRARE / IE^{IRE}

Prezentul capitol va prezenta principalele tipuri de porturi de intrare-ie^{IRE} utilizate in sisteme microprocesor de tipul 8086. Se vor prezenta atat **porturile de intrare-ie^{IRE} de uz general** de tip numeric – par.5.1, sau analogic – par.5.2), cat ^{si} cele **specializate** (seriale – par.5.3, timere – par.5.4, ^{si} controlere de intrerupere – par.5.5).

Schemele de implementare prezentate pentru porturile de I/E de uz general vor fi proiectate in ideea realizarii unei placi compatibile cu un microcalculator de tip PC-AT. In ceea ce prive^{ste} porturile specializate, acestea vor fi considerate compatibile cu cele ale unui microcalculator PC-AT, permitand ca exemplele de programe ce vor fi prezentate sa fie direct executabile pe un asemenea microcalculator.

5.1 IMPLEMENTAREA PORTURILOR DE INTRARE-IE^{IRE} NUMERICE

5.1.1 Consideratii generale privind porturile de I/E numerice

Dupa cum s-a specificat in capitolele anterioare, unitatea centrala a unui sistem microprocesor comunica cu exteriorul prin intermediul unor circuite de intrare/ie^{IRE}, numite *porturi*. Daca informatiile vehiculate de aceste circuite se prezinta la exteriorul sistemului sub forma unor **semnale binare** (contact inchis sau deschis, dispozitiv luminos aprins sau stins, semiconductor saturat sau blocat etc.), porturile respective se numesc **porturi de intrari/ie^{IRE} numerice**.

Printr-un **port de ie^{IRE} numerice**, unitatea centrala poate transmite **comenzi** (START, STOP, etc.) sau **semnalizari** (OK — starea sistemului supravegheat este normala, AVARIE — parametrul X a depa^{sit} valoarea maxima admisa) inspre exteriorul sistemului microprocesor.

Pentru a realiza aceasta, portul de ie^{IRE} respectiv va fi selectionat de catre microprocesor, iar informatia prezenta pe magistrala de date va fi "inmagazinata" de port, spre a fi disponibila la ie^{IRE}.

Selectionarea portului implica prezenta pe magistrala de adrese a unei combinatii numerice de valori binare reprezentand adresa acelui port, simultan cu activarea unor anumite semnale semnificand operatie de scriere la port, pe magistrala de control. Ele vor semnifica faptul ca, in acel moment, se realizeaza o operatie de ie^{IRE} (se executa o instructiune de tip **OUT**).

Odata selectat, portul de ie^oiri numerice face ca informatia inscrisa in el sa fie disponibila la ie^oire atata timp cat nu apare o noua selectie. Rezulta deci ca, din punct de vedere structural, un port de ie^oiri numerice are la baza un element de inmagazinare a informatiei, de memorare a ei, adica un circuit de tip *latch*. Acesta, in absenta comenzii de incarcare, i^oi mentine neschimbata starea ie^oirilor, indiferent de variatia marimilor de la intrare.

Unitatea centrala de prelucrare poate primi informatii despre starea unor marimi binare externe prin intermediul **porturilor de intrari numerice**. Selectia unui port de intrari numerice se face, ca ^oi in cazul portului de ie^oiri, pe baza adresei portului ^oi a activarii unor semnale de control (in acest caz, semnificand operatie de citire de la port). In acest caz, ele vor specifica efectuarea unei operatii de intrare (se executa o instructiune de tip **IN**). Odata selectat, portul de intrare realizeaza transferul informatiei dinspre exterior inspre magistrala de date a sistemului microprocesor. Deoarece ie^oirile unui port de intrare sunt cuplate la magistrala de date, fiecare ie^oire are la baza un circuit *tri-state*. Acesta, atunci cand nu este activat (deci portul nu este selectat), prezinta o *stare de inalta impedanta* echivalenta, functional, cu decuplarea fizica de la liniile magistralei de date.

5.1.2 Exemplu de schema de porturi de I/E numerice

Pentru a studia porturile de intrari/ie^oiri numerice ^oi analogice, se prezinta o interfata de I/E pentru un sistem microprocesor de tipul IBM-PC (compatibil 8086).

Standardul PC prevede ca spatiul de adrese pentru porturi sa fie intre 0H ^oi 3FFH. Din aceasta cauza in decodificare au fost folosite numai adresele A0 ÷ A9 (3FFH este reprezentabil pe 10 biti). Figura 5.1 prezinta schema de decodificare pentru generarea semnalelor de selectie de cip ale interfetei. Jumperii JP1, JP2, JP3, JP4, JP5, JP6 permit modificarea adresei de baza a interfetei de intrare-ie^oire.

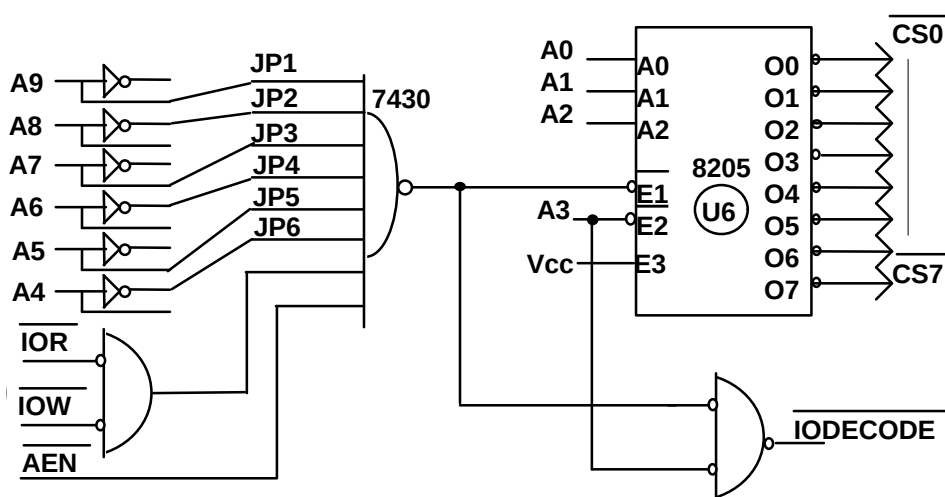


Fig.5.1. Schema de decodificare a porturilor de intrare-ie^oire de uz general

În Fig.5.1 a fost aleasă ca adresă de bază **2A0H** (JP2, JP4, JP6 - închise).

Tabela de decodificare este:

A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	∇ IOR IOW	AEN	Selectia	Adresa	Semnificatia
1	0	1	0	0	0	0	0	0	0	0	0	CS0	2A0	LED-uri
1	0	1	0	0	0	0	0	0	1	0	0	CS1	2A1	A/D
1	0	1	0	0	0	0	0	1	0	0	0	CS2	2A2	A/D
1	0	1	0	0	0	0	0	1	1	0	0	CS3	2A3	butoane
1	0	1	0	0	0	0	1	0	0	0	0	CS4	2A4	D/A octetul low
1	0	1	0	0	0	0	1	0	1	0	0	CS5	2A5	D/A octetul high

Semnalul $\overline{\text{IOCODE}}$ din Fig.5.1 indică un acces de scriere/citire de la orice port din intervalul 2A0H ÷ 2A7H. Practic nu există nici un port care să fie selectat pentru adresele 2A6H și 2A7H.

În decodificarea propriu-zisă sunt folosite numai adresele A0, A1, A2. Celelalte adrese sunt folosite pentru sintetizarea semnalului $\overline{\text{IOCODE}}$ iar A3 este folosit ca validare pentru decodificatorul U6 de tip 3 la 8. Ieșirile sale sunt active în starea low (0 logic).

5.1.2.1 Portul de ieșiri numerice

În cazul particular al exemplului de față, portul de ieșiri numerice este proiectat astfel încât valoarea semnalelor de ieșire din port să poată fi vizualizată prin starea de aprins sau stins a 8 leduri, notate LED1-LED8 (deci, un port pe 8 biți).

Portul propriu-zis este realizat în jurul circuitului 8212 (U10), utilizat ca registru *latch* cu încărcare paralelă. Semnalele de intrare în registru, D0-D7, provin de pe magistrala de date a sistemului microprocesor, octetul cel mai puțin semnificativ DB0-DB7, prin intermediul circuitului tampon 74LS245 (U4). Acesta permite transferul informației dinspre intrările A1-A8 către ieșirile B1-B8, atunci când se efectuează o operație de scriere la unul din porturile aflate în spațiul de decodificare al interfetei de I/E ($\overline{\text{IOCODE}} = 0$ și $\overline{\text{IOR}} = 1$). Urmărind schema din Fig.5.2, se observă că, dacă adresa portului la care se efectuează operația de *output* ($\overline{\text{IOW}} = 0$) va determina activarea semnalului de selecție $\overline{\text{CS0}}$, atunci informația de pe liniile D0-D7 se va încărca în circuitul 8212-U10, rămânând disponibilă la ieșirile O1-O8 ale acestuia.

Tranziția din "0" în "1" pe intrarea $DS2$ pe durata semnalului activ pe nivel "0" existent la intrarea $DS1$ determina memorarea (strobarea) datelor în latch-ul 8212-U10 și astfel disponibilitatea lor la ieșire până la următoarea comandă de încărcare.

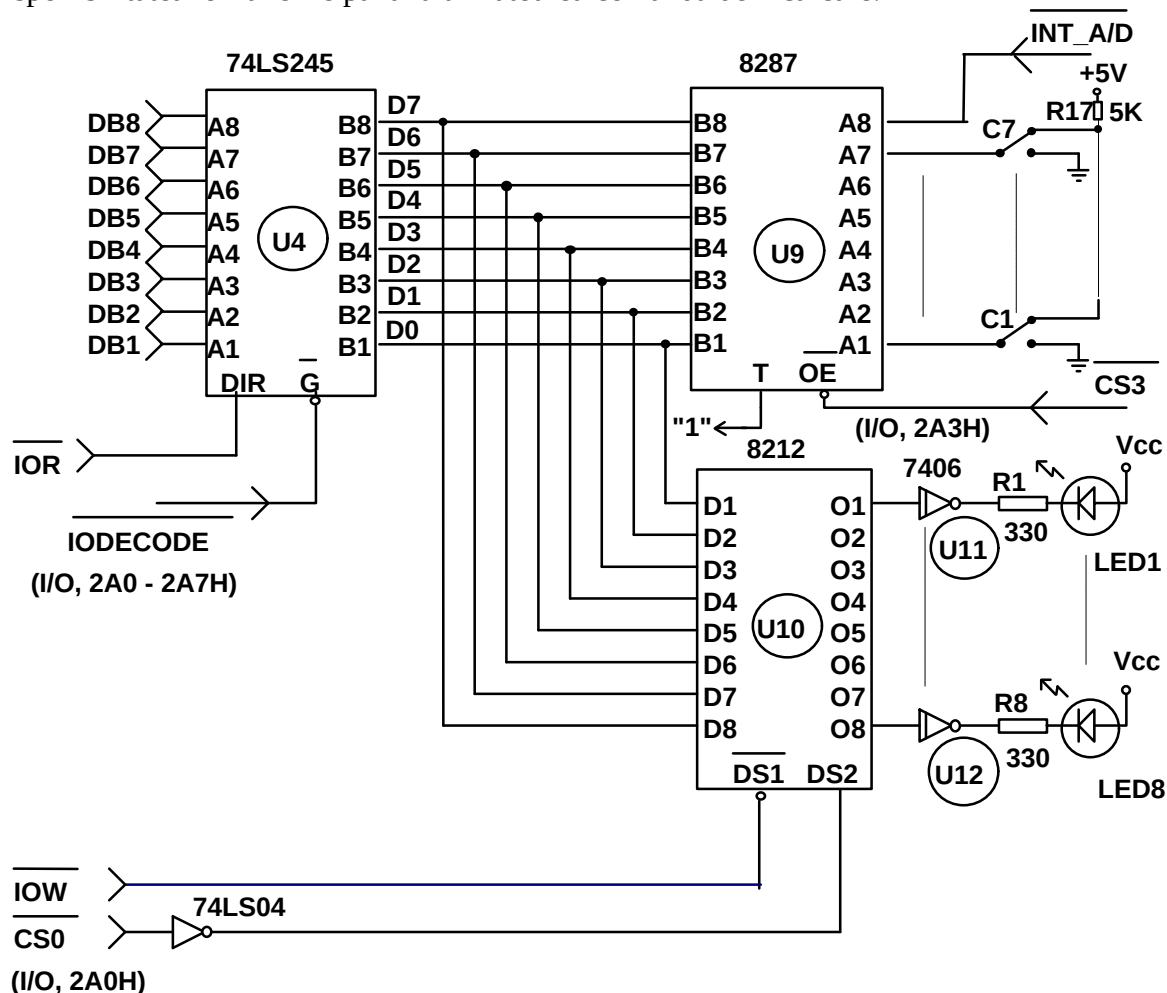


Fig 5.2. Schema porturilor de I/E numerice

Circuitele U11 și U12 de tip 7406 sunt inversoare cu nivel ridicat de *fan-out*, permitând conectarea la ieșire a ledurilor prin intermediul unor rezistențe de limitare a curentului.

În concluzie, executia unei instrucțiuni OUT la adresa 2A0H va determina aprinderea ledului j , dacă valoarea bitului ($j-1$) din octetul DB7-DB0 este 1.

5.1.2.2 Portul de intrări numerice

În schema din Fig.5.2 se implementează un singur port de intrare, utilizabil pentru citirea stării a 7 comutatoare C7-C1. Nivelurile de tensiune de 0V — comutator închis (la masă) și 5V — comutator deschis (conectat la alimentare) sunt transferate pe magistrala locală a interfetei de I/E, bitii D6-D0, prin intermediul circuitului tampon inversor 8287.

Observatii:

1. Necesitatea utilizării liniei D7 pentru a citi starea semnalului $\overline{INTA}/D$ va fi justificată în paragraful 5.2, privind porturile de I/E analogice.

Este de remarcat faptul că 8287 este un circuit tampon care se conectează la ieșire pe o magistrală de date și deci dispune de ieșiri *tri-state*, astfel încât să nu perturbe magistrala atunci când nu este selectat. El se deschide numai pe durata aplicării unui nivel logic "0" pe intrarea $\overline{OUTPUT\ ENABLE}$. Această situație apare în momentul selecției portului aflat la adresa 2A3H, când se generează prin decodificarea adresei semnalul de selecție de circuit $\overline{CS3}$. Intrarea T, de control a sensului transferului, conectată la "1" logic, comandă transfer numai în sensul "A către B".

Informația ajunsă pe liniile D0-D7 este transferată în continuare pe magistrală de date a sistemului microprocesor (DB0-DB7) prin comanda deschiderii pentru transfer "B către A" aplicată circuitului U4 ($\overline{IOR} = 0$ și $\overline{IODECODE} = 0$).

Prin urmare, executia unei instrucțiuni **IN** de la adresa 2A3H va avea ca efect transferul în microprocesor al unui octet în care bitii 0–6 reflectă starea celor 7 comutatoare (bitul **j**=1 înseamnă comutator **j**+1 închis - la masă).

5.1.2.3 Dimensionarea elementelor pasive ale scheme

Rezistența R17 conectează la alimentare comutatoarele aflate pe poziția "deschis". Ea trebuie să satisfacă cerința de a asigura curentul minim de 40 μ A necesar pentru fiecare intrare TTL a circuitului 8287, pentru nivelul logic "1".

Condiția de dimensionare este deci următoarea:

$$R < 5V / 7 \times 40\text{ mA} = 18\text{ k}\Omega$$

O valoare standard pentru această rezistență este de 5 k Ω .

Rezistențele R1-R8 conectate în serie cu ledurile se calculează din condiția de a asigura un curent optim prin leduri (10–20 mA), considerând o cadere de tensiune pe ledul în conducție de 1.2–1.5V. Totodată, curentul absorbit de o ieșire TTL (circuitul 7406) aflată pe nivel logic "0" nu poate depăși 16 mA.

Rezulta:

$$(5V - 1.5V) / 16\text{ mA} < R < (5V - 1.5V) / 10\text{ mA}$$

adică,

$$220\ \Omega < R < 350\ \Omega$$

Valoarea rezistentelor din schema este de 330 Ω .

5.1.3 Exemple de programarea porturilor de I/E numerice

În această secțiune sunt prezentate două secvențe de programare pentru utilizarea porturilor de intrare/ieșire de mai sus.

Exemplul 5.1.1. Executia secvenței de program următoare determină aprinderea ledurilor **LED7-LED1** în funcție de starea comutatoarelor **C7-C1**, astfel:

C1 închis (la masă) = se aprinde **LED1**;

....

C7 închis (la masă) = se aprinde **LED7**;

```
PORT0 EQU 2A0H    ;  
PORT3 EQU 2A3H    ;
```

```
CIT: MOV DX, PORT3 ; citește starea comutatoarelor C1-C7 de la adresa PORT0  
IN AL, DX           ; în registrul AL, bitul i=1 înseamnă comutator închis  
AND AL, 7FH         ; ledul LED8 va fi stins dacă nu există comutatorul C8  
MOV DX, PORT0       ; adresa pentru portul de ieșiri numerice  
OUT DX, AL          ; aprinde ledul i dacă comutatorul i este închis  
JMP CIT             ; repetă acțiunea
```

Exemplul 5.1.2. Secvența de mai jos prezintă o subrutină de temporizare, necesară în cazul unei aplicații cum ar fi realizarea unei "lumini dinamice":

C1 închis = se "deplasează" un led aprins dinspre **LED1** către **LED8**;

C2 închis = se "deplasează" un led aprins dinspre **LED8** către **LED1**;

C1-C7 deschise = nu se aprinde nici un led;

De această dată, efectul obținut va fi unul dinamic, cu o viteză depinzând de ceasul sistemului, ceas care are o frecvență ridicată. Din acest motiv, pentru a putea percepe evoluția în timp a deplasării luminoase a ledurilor, este necesară introducerea unei temporizări înainte sau după fiecare comandă ce schimbă starea ledurilor.

În acest scop, se poate utiliza subrutina **TAU**, exemplificată în continuare.

```
TAU:           ; subrutină de temporizare  
MOV CX, V1     ; buclă exterioară se va executa de V1 ori  
B: PUSH CX     ; salvează în stivă valoarea curentă obținută prin ;decrementarea lui V1  
MOV CX, V2     ; buclă interioară se va executa de V2 ori  
b: XCHG SI, DI ; instrucțiunea de repetat  
LOOP b         ; repetă b de V2 ori  
POP CX         ; restaurează din stivă valoarea curentă a lui V2  
LOOP B         ; repetă B de V1 ori
```

Subrutina determină executia în mod repetat a unei instrucțiuni oarecare a microprocesorului — fie aceasta **XCHG OP1, OP2** — instrucțiune care interschimbează 2 operanți. Temporizarea

dorita se realizeaza prin buclarea executiei instructiunii XCHG in doua cicluri suprapuse, obtinute prin decrementarea la zero a doua valori, V1 si V2, incarcate in registrul CX. Pentru buclare, se utilizeaza instructiunea LOOP ETICHETA, care decrementeaza valoarea din registrul CX si executa saltul la ETICHETA, daca CX>0.

Temporizarea obtinuta poate varia intre cateva microsecunde si cateva minute in functie de valorile V1 si V2 (cea mai mica intarziere se obtine daca V1=V2=1, iar cea mai mare daca V1=V2=FFFFH).

5.2 IMPLEMENTAREA PORTURILOR DE INTRARE-IE^{IRE} ANALOGICE

O interfata analog-numerica reprezinta o componenta importanta in cadrul unui sistem cu microprocesor. Marea majoritate a semnalelor provenite din mediul exterior nu au un format acceptabil pentru un sistem digital (calculator). Rolul unei interfete analog-numerice este de a transforma informatiile externe, de tip analogic, intr-un format digital si/sau invers. Exista 2 moduri de conectare a unei interfete la un sistem de calcul: prin intermediul unui canal de comunicatie standard (de tip serial sau paralel) sau direct pe magistrala sistemului.

In primul caz interfata se poate instala (fizic) langa procesul de interes, si necesita o sursa proprie de alimentare. In cel de-al doilea caz ca avantaje se pot cita: o rata mult mai mare de transfer a informatiilor intre sistemul de calcul si interfata, dimensiuni si pret de cost reduse.

In acest paragraf se prezinta un exemplu de scheme cu convertor analog-digital, respectiv digital-analog. Schemele sunt bazate, din punct de vedere al decodificarii selectiei porturilor, pe schema de selectie din Fig.5.1 din paragraful precedent.

5.2.1 Implementarea interfetei convertorului analog-digital

Este realizat ca in Fig.5.3, cu circuitul ADC0804 care are ie^{irile} de date de tip tri-state.

Plaja tensiunii de intrare este de 0 ÷ 5V. Astfel:

- pentru un semnal de intrare de 0V, codul binar al marimii convertite binar va fi 0000 0000
- pentru un semnal de intrare de 5V, codul binar al marimii convertite binar va fi 1111 1111

Data convertita este valida doar daca semnalul $\overline{INTR}$ este 0 (semnalul notat cu $\overline{INT\_A/D}$ ce intra in U9 pe bitul 7 si e citit pe magistrala de date ca bit D7 – v.Fig.5.2). Pe durata conversiei $\overline{INT\_A/D}$ este 1. Testarea acestui bit indica daca conversia s-a terminat sau nu.

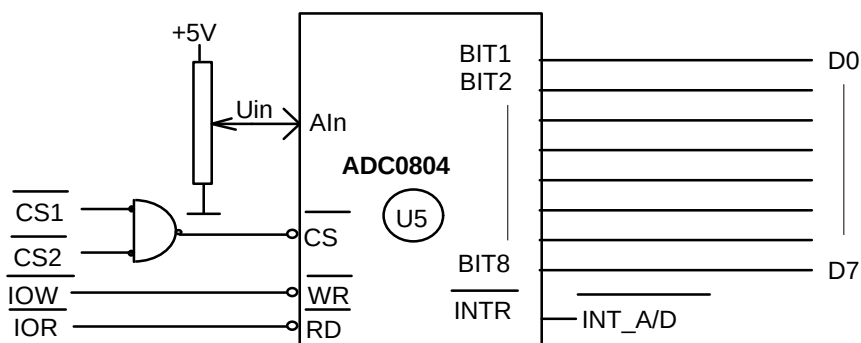


Fig.5.3. Implementarea schemei de conversie analog-numerica

Tabela de functionare a convertorului analog-digital este:

$\overline{CS}$	$\overline{WR}$	$\overline{RD}$	Funcția
1	x	x	circuit neselectat
0	1	0	citire data convertita (deschide bufferul de ieșire)
0	0	1	start conversie

Secvența de program corespunzătoare utilizării circuitului este:

```

asm
    mov dx, CS1;          Start Conversie !
    out dx,al ;
again:
    mov dx, CS3;          citire bit D7 din portul CS3
    in al,dx;
    and al,0x80;          conversia s-a terminat ?
    jz again;             daca nu așteapta

    mov dx, CS1;          conversia s-a terminat, deci:
    in al,dx;             se preia data din convertor și
    mov result, al;       se salveaza într-o variabila
end;
```

Circuitul este selectat ($\overline{CS} = 0$) pentru oricare dintre adresele 2A1H sau 2A3H ($\overline{CS} \equiv \overline{CS1} \bullet \overline{CS2}$).

5.2.2 Implementarea interfetei convertorului digital-analog

Este realizat ca în Fig.5.4, în jurul circuitului U8 (DAC 80) - convertor D/A pe 12 biți. La intrarea sa există 2 circuite latch U7 și U14 de tip Intel 8212 care păstrează intrarea digitală la ieșirile lui U8 pe durata conversiei. Acest procedeu (de a ține intrarea constantă) este necesar deoarece convertorul funcționează continuu (nu are start conversie).

Deoarece exista 2 variante de DAC 80 (cu ie^{iri} in curent sau in tensiune) a fost prevazut circuitul U13 (amplificator operational β M 741) pentru a realiza conversia din curent in tensiune in plaja $0 \div 5V$.

Datele de la intrarea convertorului sunt furnizate in 2 pa^{si}:

- 1 - se furnizeaza primii 4 biti mai semnificativi (B12 ÷ B9) printr-un OUT la adresa 2A5H (vezi schemele);
- 2 - se furnizeaza ultimii 8 biti (B8 ÷ B1) printr-un OUT la adresa 2A4H (vezi schemele).

Deoarece datele nu apar sincron (toti cei 12 biti de date) la intrarea in convertor ci cu o intarziere (dependenta de viteza procesorului) astfel: intai bitii cei mai semnificativi ^{si} apoi cei mai putin semnificativi, ie^{irea} va inregistra o fluctuatie (vizibila pe osciloscop).

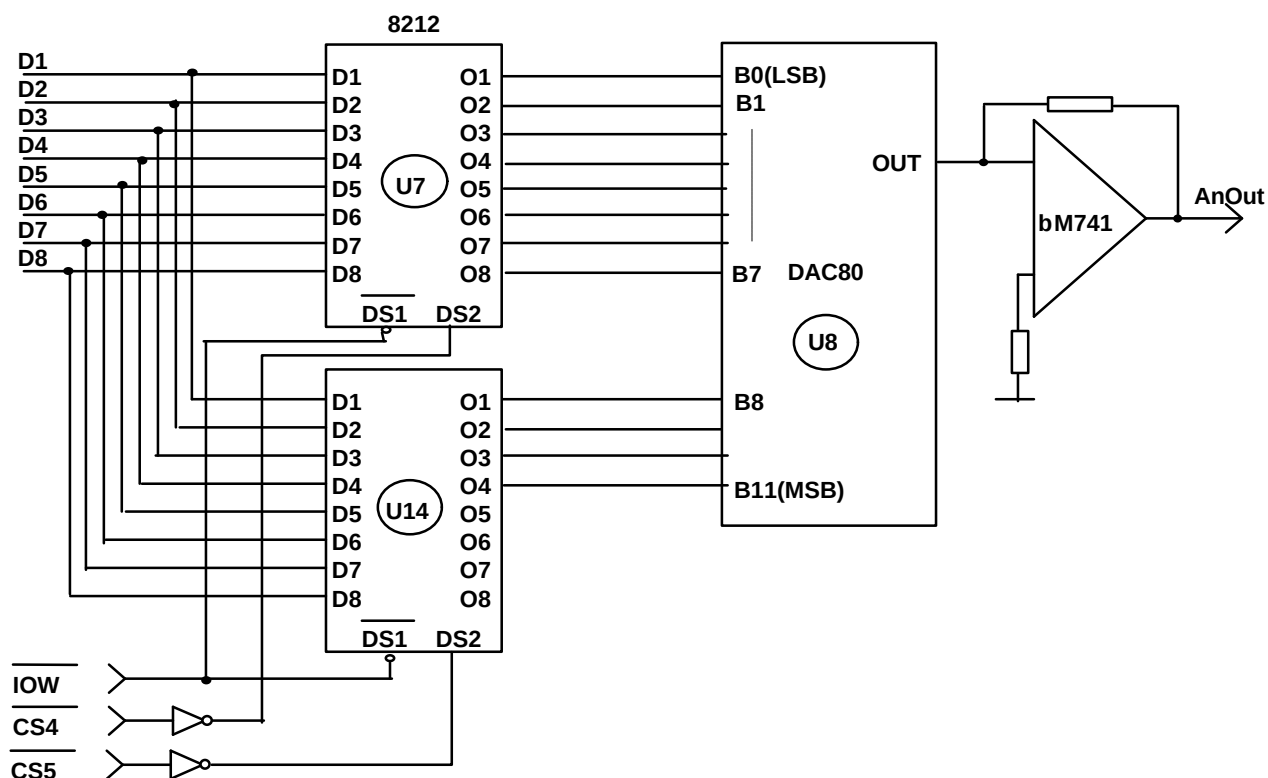


Fig.5.4. Implementarea schemei de conversie numeric-analogica

Secventa de program corespunzatoare utilizarii circuitului este:

<pre>asm mov ax, data; mov dx, CS5; out dx, ah ; mov dx, CS4; out dx, al; end;</pre>	D16 contine pe bitii B0-B11 data ce se va converti selectie port biti superiori inscriere biti superiori selectie port biti inferiori inscriere biti inferiori
--	---

5.3 PORTURI SPECIALIZATE – INTERFETE SERIALE

Dupa cum s-a precizat, porturile de I/E permit interconectarea microcalculatorului cu exteriorul, utilizind diferite **interfete de intrare-ieire**. Deoarece marea majoritate a micro-calculatoarelor vor avea anumite elemente de I/E frecvent utilizate, producatorii furnizeaza *circuite de I/E specializate*, cu un inalt grad de integrare (circuite VLSI). In aceasta categorie intra *ceasurile programabile, interfețele seriale controlerele de intreruperi, porturi paralele programabile, controlere de disc, controlere de acces direct la memorie-DMA*, etc. Se vor prezenta in paragrafele urmatoare principalele tipuri de asemenea porturi specializate, cu exemplificari concrete din familia microprocesorului INTEL 8086.

5.3.1 Interfete seriale - notiuni generale

In mod uzual, la un sistem cu microprocesor apare necesitatea schimbului de informatii – uzual date - intre sistem și utilizator, sau chiar intre doua sisteme distincte. O solutie simpla ar putea-o constitui aș-numita transmitere in paralel a datelor. De exemplu, la un sistem care are cuvantul de date de 8 biti, transmitia paralela inseamna transmiterea simultana a datelor prin intermediul unui cablu format din 8 conductori, fiecare alocat unui bit, ca in figura 5.5.

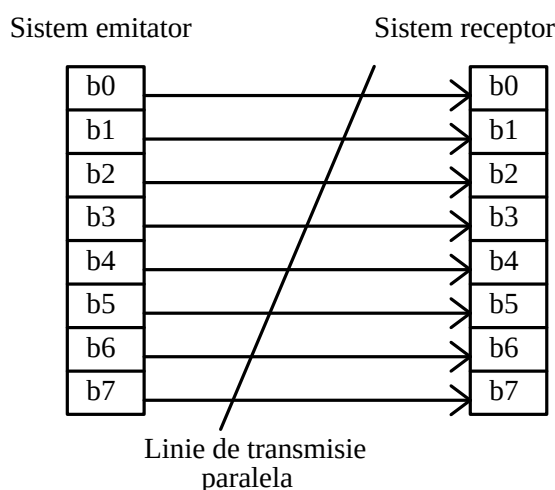


Fig.5.5. Transmisia paralela a datelor

Problema este ca transmiterea datelor intre doua calculatoare sau intre un calculator și o consola nu se poate face pe distante mai mari de cativa metri, in paralel. Motivele ar fi: problemele legate de ecranarea acestor cabluri, pretul de cost ridicat. Cu alte cuvinte, nu se pot realiza (sau nu este convenabil sa se realizeze) "magistrale" de date care sa permita transferul simultan al tuturor bitilor cuvintelor de date intre doua asemenea echipamente. Se utilizeaza in mod frecvent interfețe speciale, care transforma informatia paralela — cuvantul de date — intr-o succesiune de impulsuri, transmise pe un singur fir de legatura (vezi Fig. 5.6) in mod serial.

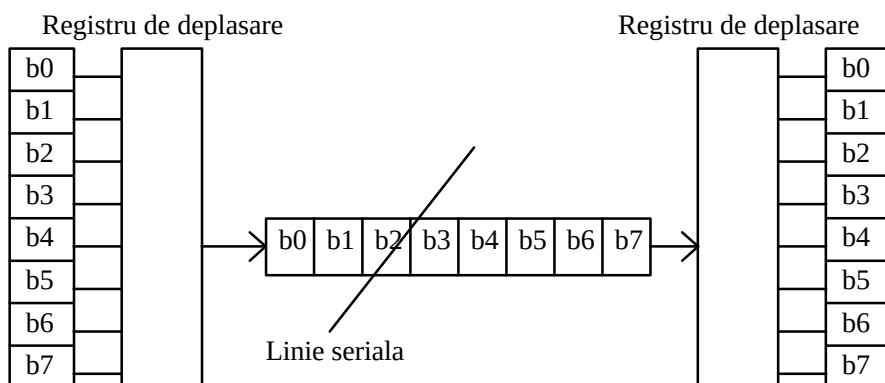


Fig.5.6. Transmisia seriala a datelor

Evident, la celalalt echipament, un circuit similar cu functionare complementara va receptiona informatia seriala, transformand-o intr-o reprezentare paralela. Deci transmisia in acest mod se face cu viteza mai mica decat transmisia paralela, nu mai apar probleme de zgomot, iar pretul echipamentelor este mult mai scazut. Trebuie remarcat ca transmisia seriala se va face prin emiterea unor impulsuri de tensiune cu doua niveluri, corespunzatoare valorilor logice "0" sau "1" ale bitilor ce se transmit. De asemenea, durata acestor pulsuri este fixa, fiind fixata de semnalul de "ceas", care va impune viteza de serializare a datelor. Este evident ca, pentru o functionare corecta, va fi necesar ca ambele circuite, atat cel emitator (transmitator) cat si receptorul, sa functioneze cu aceeași frecventa a ceasului. De aceea, aceste viteze de transmisie/receptie seriala sunt standardizate. Ele se masoara in numarul de biti ce se transmit/receptioneaza pe secunda, unitate numita *baud*. Viteze standard tipice sunt: 110, 300, 600, 1200, 2400, 4800 si 9600 de baud. Exista doua moduri principale de transmisie a datelor: modul asincron (cel mai utilizat) si modul sincron.

Structura mai detaliata a informatiei seriale transmise la unitatea unui cuvânt de date este prezentata in figura 5.7. Trebuie remarcat ca semnalul de ieșire seriala se afla in starea "1" cand nu se transmite nimic.

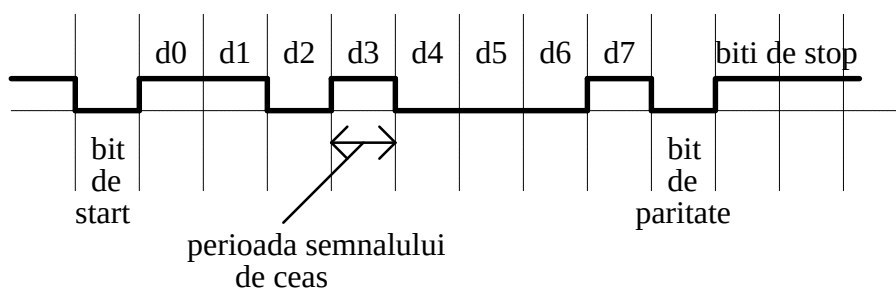


Fig.5.7. Structura informatiei seriale transmise in mod asincron

Deoarece momentul transmisiei este declanșat in mod asincron, pentru a se putea semnaliza inceperea transmisiei, intotdeauna primul bit transmis are valoarea logica "0" si se numește bit de start (vezi Fig.5.7). Acesta va declanșea, la receptor, operatia de receptie si deserializare a urmatoarelor biti, ce reprezinta informatia propriu-zisa care se transmite (b0 pana la b7 in figura 5.7). Deasemeni, din ratiuni ce tin de viteza de lucru a acestor echipamente, transmisia se va

incheia cu un numar precizat (cel putin 1) de biti de stop (valoarea logica "1"). Abia apoi se va putea emite, eventual, un alt cuvânt de date, in aceeași succesiune a operatiilor.

Observatii

1. In figura 5.7, cuvântul de date transmis corespunde valorii binare

1 0 0 0 1 0 1 1 sau hexazecimal 8BH
(bit7) (bit0)

2. Se observa ca informatia "utila" este formata din 8 biti (d0-d7), ceilalti 4 biti (start, paritate + 2 biti de stop), "consumand" din timpul de transmisie și facand ca acest protocol (mod) sa nu fie prea eficient. O posibilitate de imbunatatire a acestuia ar putea sa para "lungirea" cuvântului de date. Trebuie spus insa ca, la acest protocol, atat receptorul cat și emitorul au propriul lor cuvânt de ceas. Chiar daca ele sunt programate sa functioneze la aceeași viteza, practic dupa o anumita perioada de timp ele se desincronizeaza, fapt ce conduce la aparitia erorilor de transmisie. S-a demonstrat statistic ca o lungime de 8 biti a cuvântului de date este optima pentru acest protocol.
3. Pentru o mai buna sincronizare a transmisiei/receptiei, se utilizeaza uneori un semnal de ceas al circuitului de serializare, cu o frecventa mai mare decat cea de transmisie/receptie — uzual, de 16 sau 64 de ori, divizarea suplimentara facandu-se in circuitul de serializare.
4. Intr-un mod oarecum diferit se efectueaza operatiile in așa-numitul mod sincron de lucru, posibil cu unele circuite de interfata seriala. Principial insa, lucrurile sunt similare, doar protocolul dialogului intre transmitator și receptor este mai complicat. Principala diferenta consta in aparitia unei linii suplimentare care poarta informatia de ceas, furnizata de obicei de transmitator. Aceasta face ca atat receptorul, cat și transmitatorul sa functioneze cu același ceas, deci sa se reduca erorile de transmisie. Natural, aparitia unei linii suplimentare marește probabilitatea de aparitie a zgomotului, cat și costul.
5. Este frecvent utilizat, inaintea bitului de stop, un "bit de paritate", care permite verificarea corectitudinii transmiterii informatiei. Astfel, pentru "paritate para", suma modulo 2 a tuturor bitilor transmiși (b0 pana la b7) și a bitului de paritate trebuie sa dea "0". Similar, in cazul "paritatii impare", aceasta suma trebuie sa dea "1". Neconcordantele se semnaleaza; uzual, circuitul serializator inverseaza bitul de paritate/analizeaza corectitudinea paritatii la receptie.
6. Cuvintele de date vehiculate serial pot contine informatii binare interne ale calculatoarelor. Totuși, uzuala este vehicularea seriala a informatiilor codificate in așa-numitul cod ASCII (*American Standard Code of Information Interchange*), in care toate caracterele alfanumerice tiparibile și o serie de coduri standardizate de control au un cod corespunzator. Utilizand 7 biti in versiunea standard, codul ASCII are deci 128 de componente; in versiunea extinsa, codul ASCII utilizeaza cuvinte de 8 biti, in numar de 256.

De exemplu, litera "A" are codul ASCII 01000001B (binar) sau 41H (hexa), iar cifra "5" are codul ASCII 00110101B (binar) sau 35H (hexa), etc.

7. Legatura seriala se face la alte niveluri de tensiune decat cele TTL. Uzual, se folosesc:
- (a) "bucle de tensiune" (0-12V) pentru legaturi pana la 200 m;
 - (b) "bucle de curent" (0-20mA) pentru legaturi pana la 2 km;
 - (c) Circuite "modem" pentru modularea semnalelor, la transmisii pe distante mari (linii telefonice).
 - (d) Standardul RS-232 prevede pentru nivelul logic "1" o tensiune de -12V, cu un nivel limita de -3V, iar pentru nivelul logic "0" o tensiune de +12V, cu un nivel limita de +3V, pentru distante de ordinul kilometrilor.

5.3.2 Interfata seriala INTEL 8251

In familia circuitelor INTEL, se utilizeaza ca interfata seriala programabila circuitul INTEL 8251. Circuitul poate lucra atat in mod sincron, cat ^oi asincron, permitand atat transmisia, cat ^oi receptia seriala. (Denumirea sa: circuit USART = *Universal Synchronous Asynchronous Receiver Transmitter*). Este un circuit LSI, cu multe facilitati.

Vom prezenta in continuare doar acele elemente necesare pentru utilizarea circuitului in mod asincron. Pentru detalii suplimentare, se va consulta catalogul firmei.

Circuitul contine 2 porturi de intrare ^oi 2 porturi de ie^{ire}, corespunzatoare cuvintelor de control/date de intrare ^oi ie^{ire}. Le vom nota cu IUS1, IUS2, respectiv OUS1, OUS2.

(a) Cuvinte de control

Cuvantul 1 de comanda — mod asincron (OUTPUT, la portul OUS2)

b7 b6 b5 b4 b3 b2 b1 b0

-----	-----	-----	-----	
				00 - invalid
				01 - rata x 1
				10 - rata x 16
				11 - rata x 64
				00 - car. 5 biti
				01 - car. 6 biti
				10 - car. 7 biti
				11 - car. 8 biti
				x0 - dezactivare
				01 - paritate para
				11 - paritate impara
				00 - invalid
				01 - 1 bit de stop
				10 - 1 1/2 bit de stop
				11 - 2 biti de stop

Cuvantul 2 de comanda — mod asincron (OUTPUT la portul OUS2)

b7	b6	b5	b4	b3	b2	b1	b0
0	0	1	1	0	1	1	1

- initializeaza circuitul
- pentru detalii, vezi manual firma

Cuvantul de stare (INPUT, de la portul IUS2)

b7	b6	b5	b4	b3	b2	b1	b0
-----							-- TxRdy (transmisie gata)
							----- RxRdy (receptie gata)

alte conditii (vezi catalogul)

(b) **Cuvinte de date** (porturile OUS1, IUS1)

Dupa programarea circuitului (printr-o instructiune de OUTPUT la portul OUS2), se pot transmite serial caractere (output la portul OUS1), sau citi — daca s-au receptionat — caractere (input de la portul IUS1).

Orice operatie de transmisie se va face daca s-a terminat transmisia unui caracter anterior. Pentru aceasta, se va transmite circuitului 8251 data doar dupa ce se testeaza starea transmitatorului (vezi cuvantul de stare, bitul 1). Astfel, se va aștepta incheierea acestei transmisii și doar apoi se va continua, cu un nou caracter (doar daca bitul 1 din cuvantul de stare are valoarea 1, indicand incheierea transmisiei anterioare). Similar, se citește o data valida prin INPUT de la adresa IUS1 doar daca, in cuvantul de stare (citit cu INPUT de la adresa IUS2), bitul indicand receptia unui caracter (bitul2) este setat la valoarea 1. Altfel, se aștepta setarea acestui bit, dupa care se face citirea datei convertite serial-paralel.

Observatii

1. Dupa cum s-a precizat, este imperios necesar ca rata de transmisie a circuitului emitor sa fie egala cu rata de receptie a circuitului receptor. In caz contrar, informatia este eronat reconstituita la receptor.
2. Se recomanda alimentarea circuitului cu un semnal de ceas de 16 (sau 64) de ori mai mare decat rata de transmisie dorita și setarea corespunzatoare a cuvântului de comanda. De exemplu, pentru a obtine o rata de transmisie de 300 de baud, este necesar:
 - fara divizare suplimentara: ceasul trebuie sa aiba frecventa 300 Hz
 - cu divizare x16: ceasul trebuie sa aiba frecventa 16x300=4.800 Hz
 - cu divizare x64: ceasul trebuie sa aiba frecventa 64x300=19.800 Hz

3. Legarea seriala standard a doua echipamente se va face, uzual, prin intermediul a 3 conductoare de legatura, ca in Fig.5.8.

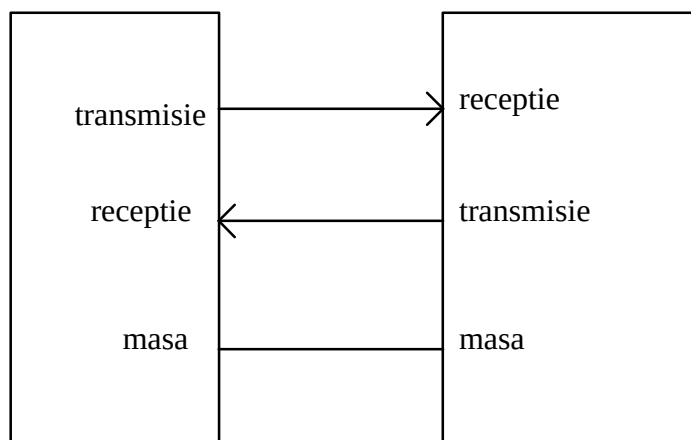


Fig.5.8. Conectarea seriala a doua echipamente

4. Se pot transmite cuvinte de date nu doar de 8 biti (tipic), ci și de 7, 6, 5 sau chiar 4 biti.

5.3.3 Porturile de comunicatie seriala la calculatoarele de tip PC-AT

Marea majoritate a calculatoarelor PC-AT compatibile dispun de un sistem de comunicatie seriala construit in jurul unui circuit INTEL 8251, sau al unuia compatibil. Pentru a simplifica programarea acestuia, o serie din logica de comanda a fost scoasa pe niște registre accesibile, prin operatii de intrare/ieșire pe porturi de 8 biti. Numele și adresa de port ale acestor registre sunt date in tabelul 5.1.

Tabelul 5.1.

Adresa	Registrul	DLAB
XF8	Registrul tampon TX de transmisie date	0 (scriere)
XF8	Registrul tampon RX de receptie date	0 (citire)
XF8	Partea Low a Baud generator LBG	1
XF9	Partea High a Baud generator HBG	1
XF9	Registrul de activare a intreruperilor	0
XFA	Registrul de identificare a intreruperilor	
XFB	Registrul de control linie RCL	
XFC	Registrul de control modem	
XFD	Registrul de stare linie RSL	
XFE	Registrul de stare modem	

Observatii:

1. Litera X din adresele registrelor va fi:
X = 3, in cazul in care ne referim la primul port serial (COM1)

$X = 2$, in cazul in care ne referim la primul port serial (COM2)
Uzual se va lucra pe al doilea port serial, primul fiind utilizat pentru *mouse*.

2. Din tabelul de mai sus vor fi detaliate aici registrele cele mai utilizate. Pentru mai multe informatii, se va consulta catalogul firmei.
3. A^o cum se constata, exista situatii in care mai multe registre sunt plasate la aceea^oi adresa (XF8, ^oi XF9). Acest lucru este posibil prin faptul ca registrele in cauza nu sunt folosite niciodata simultan. Controlul asupra accesului la aceste registre se face prin intermediul bitului 7 (DLAB) din registrul de control linie ^oi prin operatia ce se executa asupra registrului. Astfel, daca bitul DLAB este 0, atunci o citire a portului XF8 ne da valoarea registrului RX (de receptie), iar o scriere in acela^oi port va scrie in registrul TX (de transmisie). Daca bitul DLAB este 1, atunci la adresele XF8 ^oi XF9 se vor scrie valori ce vor determina rata de transmisie/receptie in Baud.

Registrele TX ^oi RX sunt registre ce vor contine cuvantul transmis, respectiv ultimul cuvant de date receptionat.

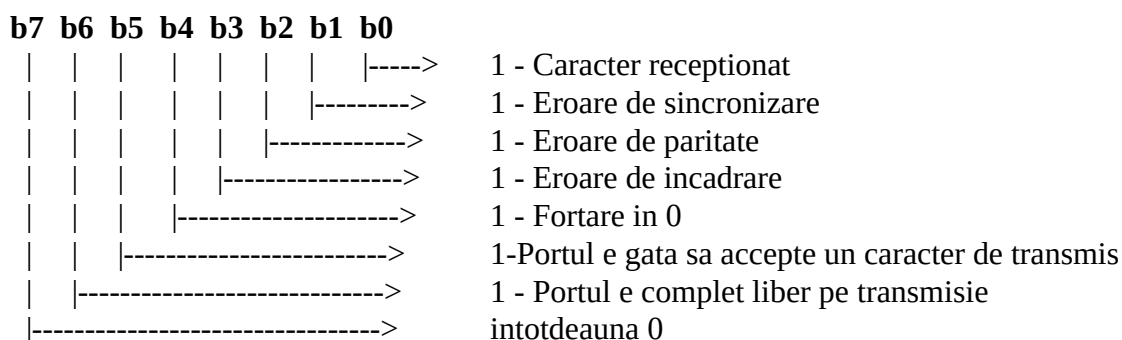
Registrele LBG ^oi HBG sunt folosite la programarea vitezei de transmisie. Ele vor fi scrise o singura data, la initializarea transmisiei, ^oi apoi nu vor mai fi accesate. Pentru accesul lor, se va seta pe nivel logic "1" bitul 7 din registrul RCL. Pentru viteza de 9600 baud, ele vor contine:

LGB = (0CH), iar HGB = (0H). Pentru mai multe informatii, se va consulta catalogul firmei.

Registrul RCL este registrul prin care comandam efectiv functionarea portului serial. Semnificatia bitilor sai este urmatoarea:

b7	b6	b5	b4	b3	b2	b1	b0	

							----->	Lungimea cuvantului de date:
								00 - 5 biti
								01 - 6 biti
								10 - 7 biti
								11 - 8 biti
						----->		Numarul de biti de stop: 0 - 1 bit de stop
								1 - 2 bit de stop
				----->				Activeaza verificarea paritatii
			----->					Selecteaza paritatea para (uzual 0)
		----->						Blocheaza paritatea (uzual 0) — neutilizat
	----->							Forțeaza 0 pe transmisie
----->								DLAB (controleaza accesul la setare Baud sau registrele de transmisie/receptie (vezi obs. 3 de mai sus)



Restul regi^otrilor sunt folositi pentru comunicatia prin modem (linie telefonica) ^oi pentru lucrul in intreruperi al portului.

Se prezinta in continuare listingul complet al unui exemplu de programare a portului serial COM2 al unui microcalculator compatibil de tip PC-AT. Se prezinta rutina de initializare a comunicatiei seriale la o viteza de 9600 baud, pe cuvinte de 8 biti. Se definesc deasemenea ρ cele doua rutine de baza necesare pentru a implementa comunicatia seriala, rutina de transmisie ρ rutina de receptie a unui caracter la / de la interfata seriala.

Exemplu de programare a portului serial COM2:

```
; Subrutina de initializare a portului pentru 9600 baud, cuvinte 8 biti, fara controlul paritatii,
; un bit de stop
```

initser:

EQU	COMD2	0x2f8	; adresa portului de date com2
EQU	COMC2	0x2fB	; adersa RCL
EQU	COMS2	0x2fD	; adresa RSL
EQU	COML2	0x2f8	; adresa LBG
EQU	COMH2	0x2f9	; adresa HBG

```
MOV DX,COMC2
MOV AL,$80
OUT DX,AL           ; aici se seteaza DLAB=1 pentru programare Baud rate
MOV DX,COML2
MOV AL,$0C
OUT DX,AL
MOV DX,COMH2
MOV AL,0
OUT DX,AL           ; Baud rate = 9600
MOV DX,COMC2
MOV AL,3             ; 8 biti, fara paritate, 1 bit de stop
```

```
OUT DX,AL
```

```
RET
```

```
;aici s-a terminat initializarea
```

```
;*****
```

```
; Rutina de transmisie a unui caracter aflat in registrul AH
```

```
co:
```

```
    MOV DX,COMS2
```

```
    IN AL,DX           ; citește starea portului
```

```
    AND AL,$20         ; este gata de transmisie? (bitul 5 e pe 1?)
```

```
    JZ  co             ; daca nu, continua sa incerci
```

```
    MOV DX,COMD2       ; daca da
```

```
    MOV AL,AH
```

```
    OUT DX,AL          ; transmite caracterul
```

```
; *****
```

```
; Rutina de receptie a unui caracter. Se va receptiona caracterul in registrul AL
```

```
ci:
```

```
    MOV DX,COMS2
```

```
    IN AL,DX           ; citește starea portului
```

```
    AND AL,1           ; a receptionat ceva?
```

```
    JZ  ci             ; daca nu, continua sa incerci
```

```
    MOV DX,COMD2       ; daca da,
```

```
    IN AL,DX           ; preia in AL caracterul receptionat
```

```
; *****
```

5.4 PORTURI SPECIALIZATE – CEASURI DE TIMP REAL

5.4.1 Ceasuri programabile – notiuni generale

Functionarea microcalculatoarelor in aplicatii de proces va necesita, in majoritatea cazurilor, realizarea unor temporizari precise intre diversele actiuni efectuate in sistem. In acest scop, se utilizeaza a^oa-numitele **ceasuri de timp real** (*Real Time Clock* — **RTC**), circuite ce permit obtinerea unor temporizari programate, cu precizie ridicata. In principiu, aceste circuite vor diviza un semnal de intrare de frecventa fixa, cu un numar precizat prin programare, functionand ca numaratoare programabile (furnizand un semnal cu factor de umplere apropiat de 0.5, deci aproximativ jumatate din perioada semnalului de ieșire are valoare logica 1, iar cealalta jumatate de perioada are valoare logica 0; vezi Fig 5.9.a) sau ca circuite de temporizare (furnizand un puls, de durata egala cu cea a semnalului de ceas, dupa o perioada de timp egala cu N perioade de ceas; v. Fig. 5.9.b). Uzual, ieșirea *RTC* se conecteaza la sistemul de intreruperi al microcalculatorului, declanșind lansarea unei subrutine specifice la sfarșitul temporizarii respective. *Semnalul de intrare* al *RTC* poate fi semnalul de ceas al sistemului, dar poate proveni și din exterior (de exemplu, impulsurile de la un traductor incremental). Uneori este posibila sincronizarea declanșarii numararii de la un semnal exterior .

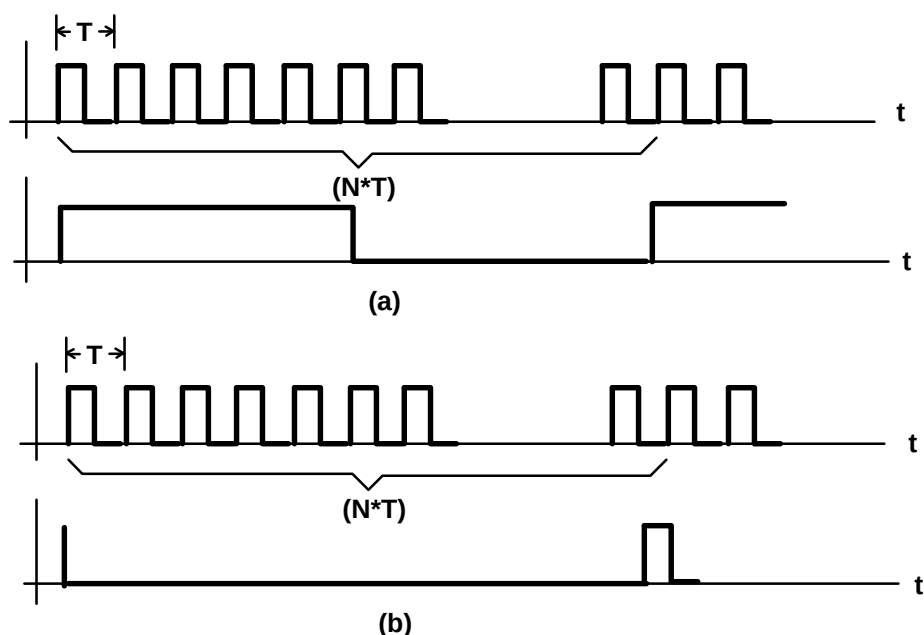


Fig.5.9. Principalele moduri de numarare ale unui circuit de tip RTC:
 (a) numarator programabil; (b) ceas programabil

5.4.2 Ceasul programabil INTEL 8253

În familia circuitelor firmei *INTEL*, se utilizează ca *circuit de ceas programabil* circuitul **INTEL 8253**. Acesta este un circuit VLSI conținând 3 *numaratoare de câte 16 biți* (fiecare putând, deci, să divida semnalul de intrare până la de $2^{16} = 65535$ ori). Dacă se dorește o divizare și mai mare a frecvenței semnalului de intrare, se pot lega în cascada două, sau chiar toate trei numaratoarele circuitului. Prin programare (utilizând cuvinte de comandă și de date corespunzătoare) se indică modul de lucru al circuitului (sunt posibile 5 moduri distincte) și se programează numaratoarele utilizate.

Circuitul 8253 operează cu cuvinte de 8 biți, fiind direct compatibil cu microprocesoarele 8080, 8085, eventual Z80. Este, de asemenea, utilizat în microcalculatoarele de tip IBM PC cu microprocesor 8086 etc.

Din punct de vedere al adresării, circuitul **8253** ocupă un număr de **4 adrese de I/E**, deci conține, practic, din punct de vedere funcțional, 4 porturi de intrare și 4 porturi de ieșire.

Observatie: Porturile de intrare și cele de ieșire ale circuitului 8253 au aceleași adrese. Nu este obligatorie nici o legătură între două porturi, unul de intrare și altul de ieșire, având aceeași adresă; ele sunt differentiate tocmai de semnalele de control ce indică sensul transferului datelor: *citire* sau *scriere*.

Figura 5.10 prezintă schema de principiu a circuitului INTEL 8253. După cum se poate remarca, circuitul 8253 conține un registru de control, care coordonează funcționarea celor 3 numaratoare de câte 16 biți fiecare, incluse în circuit.

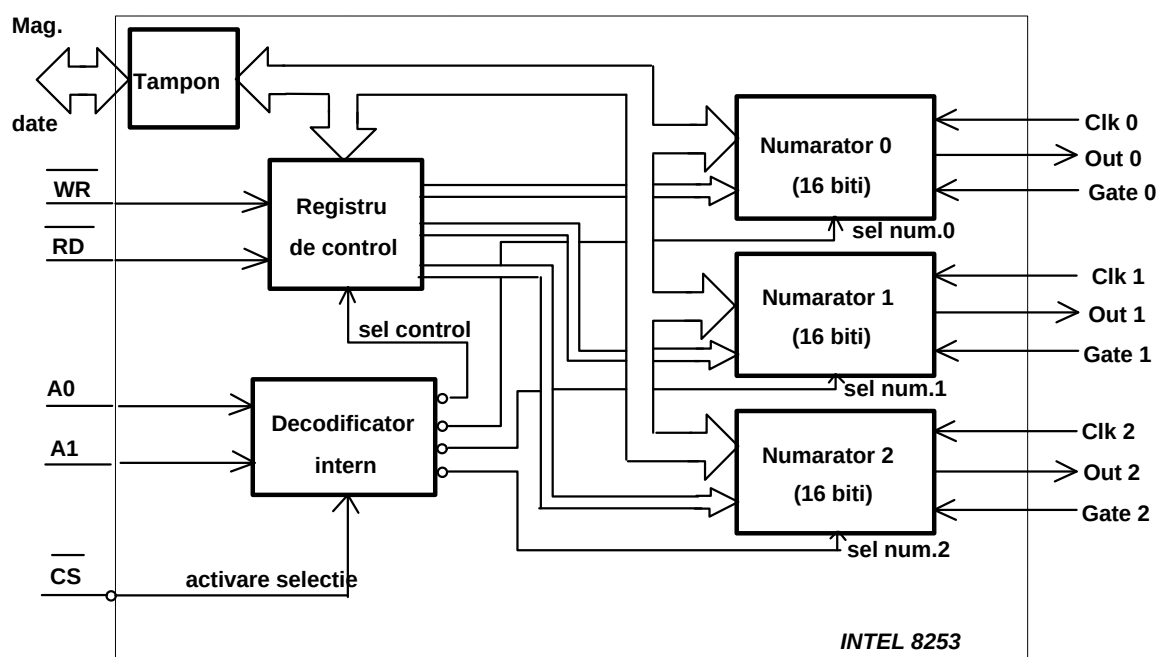


Fig.5.10. Structura de principiu a circuitului INTEL 8253

Decodificatorul intern utilizează doi biti de adresă ce trebuie furnizați circuitului 8253, pentru a selecta unic unul din cele 4 porturi de I/E ale acestuia: *portul de control* și cele *trei porturi* corespunzătoare celor *trei numărătoare*. Pentru *portul de control*, o operație de *scriere* va determina înregistrarea în registrul de control a unui *cuvânt de comandă*, indicând ce operație și asupra cărui numărator se va efectua în continuare. O operație de *citire de la portul de control nu este permisă* (vezi aliniatul următor pentru detalii). *Scrierea* la un port corespunzător unuia dintre *numărătoare* va determina *programarea* acestuia cu valoarea *intervalului de numărare* (numărul de pulsuri de ceas pe care le va număra număratorul respectiv). *Citirea* de la portul corespunzător unuia dintre aceste *numărătoare* va determina citirea valorii curente a *conținutului* număratorului respectiv. Numărarea are loc prin scăderea conținutului număratorului, cu câte o unitate la fiecare impuls de ceas primit pe intrarea sa de ceas (Clk_i).

Logica de control a circuitului se bazează pe utilizarea semnalelor $\overline{CS}$, $\overline{RD}$, $\overline{WR}$, $A0$ și $A1$. Semnalul de $\overline{CS}$ (*Chip Select* — selecție de cip) validează funcționarea circuitului (fără ca acest semnal să fie activ = 0 logic, circuitul nu este selectat și funcționarea sa nu este afectată). Semnalul $\overline{RD}$ activ indică citirea din registrul de control sau din număratoarele, iar semnalul $\overline{WR}$ activ asigură încărcarea cuvântului de control sau a datelor în număratoarele. Cei doi biti de adresă, $A0$ și $A1$, vor selecta, după cum s-a menționat, unul din cele 4 porturi ale circuitului. Tabela 5.2 sintetizează funcționarea circuitului, în funcție de valorile acestor semnale.

Semnalele $Clk0$, $Clk1$, respectiv $Clk2$ vor furniza semnalul de intrare, care va fi divizat prin numărare, pentru fiecare dintre cele trei număratoarele (0, 1 și, respectiv, 2). Semnalele $Gate0$, $Gate1$ și $Gate2$ vor putea fi utilizate pentru sincronizarea circuitului cu exteriorul (ele pot bloca număratorul asociat). Semnalele $Out0$, $Out1$ și $Out2$ reprezintă ieșirile celor trei

11 - ilegal

Dupa cum se poate remarca, pentru a programa circuitul 8253 trebuie efectuata o operatie de inscriere a unui *cuvant de comanda* (*OUTPUT* la portul OTIM4), care specifica:

- ce numarator se va programa (bitii 6 și 7 din cuvantul de comanda);
- in ce mod va lucra numaratorul (bitii 1, 2, 3); modurile au urmatoarea semnificatie:
 - modul 0 = intrerupere, la terminarea numararii;
 - modul 1 = monostabil programabil;
 - modul 2 = generator de impulsuri divizate cu N;
 - modul 3 = generator de impulsuri dreptunghiulare;
 - modul 4 = strob comandat prin software;
 - modul 5 = strob comandat prin hardware;
- in ce mod se va incarca valoarea numerica (16 biti) reprezentand cu cat va divide numaratorul respectiv (bit 4 și 5);
- reprezentarea numerica a acestui numar: binar sau BCD (bitul 0).

Cuvintele de date (*OUTPUT* — date de ieșire) se utilizeaza dupa ce s-a dat cuvantul de comanda pentru numaratorul care a fost selectat prin cuvantul de comanda. Permit inscrierea valorii numerice cu care va divide numaratorul respectiv. Se utilizeaza:

OUTPUT la portul OTIM1, pentru programarea numaratorului 1

OUTPUT la portul OTIM2, pentru programarea numaratorului 2

OUTPUT la portul OTIM3, pentru programarea numaratorului 3

Cuvintele de date (*INPUT* — date de intrare) permit citirea continutului numaratoarelor circuitului 8253, la un moment dat.

(**Obs.:** Deoarece uzual nu sunt utilizate, informatii suplimentare despre toate modurile de lucru, cuvantul de stare și cuvintele de date de intrare se pot gasi in datele de catalog ale circuitului).

Un exemplu de programare al circuitului 8253 se prezinta in sectiunea 5.1.4 a lucrarii.

5.4.3 Ceasurile programabile la calculatoarele de tip PC-AT

In prezentul paragraf se prezinta un exemplu de utilizare al ceasului programabil existent pe microcalculatoarele de tip PC-AT și compatibile. Se va utiliza circuitul 8253 existent in sistem, și anume numaratorul 2, disponibil pentru a putea fi programat de catre utilizator, care poate alimenta difuzorul sistemului. Totodata, deși nu se va intra in detalii aici, se utilizeaza și doi dintre bitii de date ai unui circuit de tip port paralel programabil, 8255, pentru a se permite accesul și, respectiv, blocarea semnalului de ieșire al numaratorului 2 al circuitului 8253, catre difuzorul microcalculatorului. Schema de principiu a acestei parti a sistemului este prezentata in figura 5.11.

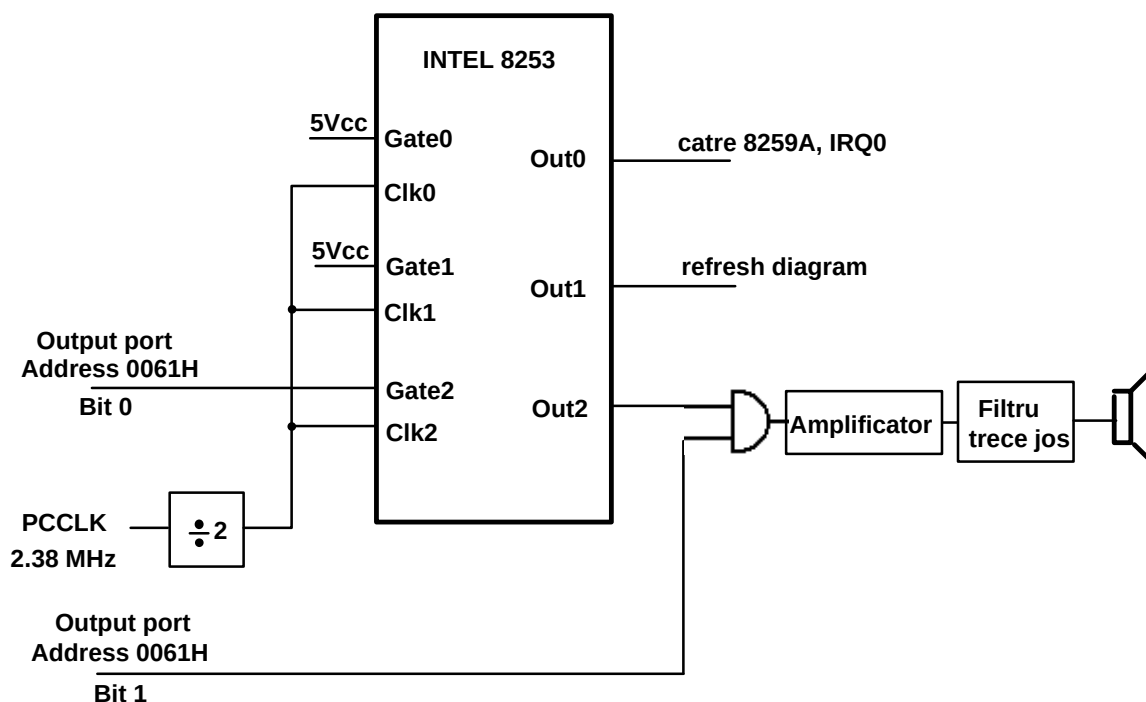


Fig.5.11. Schema de conexiuni externe a circuitului 8253, de pe microcalculatoarele de tip PC-AT

Porturile circuitului de ceas programabil 8253 ocupa, in spatiul de **adrese de I/O** ale microcalculatorului PC realizat cu microprocesorul 8086 (sau cu microprocesoarele compatibile: 80286, 80386, 80486):

- ITIM0, OTIM0: 40H (furnizeaza tactul pentru ceasul de sistem al microcalculatorului);
- ITIM1, OTIM1: 41H (furnizeaza semnalul de reamprospatare a memorie dinamice pentru circuitul DMA, canalul 0, al sistemului);
- ITIM2, OTIM2: 42H (furnizeaza semnalul pentru difuzorul sistemului);
- OTIMC: 43H (registrul de comanda al circuitului).

Observatii:

1. Calculatoarele de tip PC-AT și cele bazate pe microprocesorul 80386, 80486 utilizeaza circuitul 8254 — similar, in cele prezentate, cu circuitul 8253.
2. Adresele din zona de porturi de la 44H pana la 5FH sunt decodificate redundant, pentru circuitele 8253 / 8254 ale sistemului.

Semnalul de intrare al numaratorului 2 al circuitului 8253 (8254) este, dupa cum se remarca din figura 5, de 1.19 MHz. Aceasta frecventa de baza trebuie considerata pentru determinarea valorii de divizare, pentru a genera la ieșirea numaratorului un semnal de frecventa dorita.

Deoarece numaratorul poate fi programat sa divida cu minim 2 și maxim 65535, plaja de frecventa a semnalului de ieșire al numaratorului va fi:

- maximum 595 KHz (divizare cu 2);

- minimum 18,15 Hz (divizare cu 65535).

Trebuie remarcat ca validarea numararii va fi posibila prin activarea a doi biti suplimentari ai portului paralel 8255 al sistemului, de la adresa 61H, ºi anume bitii 0 ºi 1 ai acestui port. Pentru ca numaratorul sa numere, trebuie setat bitul 0 al acestui port, iar pentru ca semnalul de ieºire al numaratorului sa fie efectiv aplicat catre difuzor, trebuie setat bitul 1 al portului paralel de ieºire de la adresa 61H.

Observatie importanta: Dupa cum se constata ºi din schema din Fig. 5, celelalte numaratoare ale circuitului 8253 sunt folosite de catre sistem pentru alte resurse ale sistemului. Aceeaºi situatie exista ºi pentru circuitul 8255, ceilalti biti ai portului de ieºire de la adresa 61H fiind utilizati pentru controlul altor resurse ale sistemului. **IN CONSECINTA**, este imperios necesar ca, in aplicatiile ce se vor testa in continuare, sa **NU SE MODIFICE** (prin programare, sau date eronate) valorile de la celelalte numaratoare ale circuitului 8253, sau bitii 2–7 de pe portul de ieºire de la adresa 61H.

Rezumand, adresele porturilor ce se vor utiliza practic, pentru a determina (re)programarea numaratorului 2 al circuitului 8253 ºi generarea unui sunet la difuzorul sistemului, sunt:

port	adresa	tip port	functie
OTIMC	43H	OUTPUT	port control
OTIM3	42H	OUTPUT	inscriere valoare dividere numarator
8255, canal B	61H	OUTPUT	bitii 0,1: setati la 1 = validare ieºire 8253; bitii 0,1: setati la 0 = blocare ieºire 8253.

5.4.4 Exemplu de utilizare a ceasurilor programabile la calculatoarele de tip IBM-PC

Vom prezenta trei subrutine care realizeaza:

- programarea numaratorului 2 al circuitului 8253 pentru a genera un semnal de frecventa data;
- programarea circuitului 8255, canalul B (portul de ieºire de la adresa 61H), pentru a debloca semnalul generat de catre numaratorul 2 al circuitului 8253, catre difuzorul sistemului;
- programarea circuitului 8255, canalul B (portul de ieºire de la adresa 61H), pentru a bloca semnalul generat de catre numaratorul 2 al circuitului 8253, catre difuzorul sistemului.

Pentru obtinerea unui semnal de 500 Hz, cuanta de divizare a numaratorului va fi:

$$N = f_{in} / f_{out} = 1\,193\,180 / 500 = 2386 = 0952H$$

Modul cel mai convenabil de numarare al numaratorului, pentru a genera un semnal sonor, este modul 3 (semnal dreptunghiular, vezi par. precedent).

Cu aceste observatii, cele trei subrutine sunt prezentate in continuare:

```

=====
;;      Subrutina pentru initializarea circuitului 8253
;;      -----
;;      Argumente: <Intrare>: BX = valoarea N, cu care va divide numarul 2 al 8253
;;                  <leire> : -
;;      Altereaza:   registrul AX
;;
OTIMC   EQU    43H      ; adresa portului de control al 8253
OTIMD   EQU    42H      ; adresa portului de date al numaratorului 2 al 8253
DTCMD   EQU    0B6H     ; valoarea cuvintului de comanda al 8253 = 1011 0110 b, deci:
                        ; set numarator 2; mod 3 de numarare; incarcare succesiva
                        ; (octet low, apoi high); numar incarcat in reprezentare binara
;
INITIM2: MOV     AL,     DTCMD ; incarca AL cu cuvintul de comanda
          OUT     OTIMC, AL    ; transmite cuvintul de comanda la 8253
          MOV     AX,     BX    ; incarca AX cu valoarea N
          OUT     OTIM2, AL    ; transmite low(N) la 8253, numaratorul 2
          OUT     OTIM2, AH    ; transmite high(N) la 8253, numaratorul 2
;
          RET          ; programarea incheiata, return
=====
;;      Subrutina pentru activarea sunetului la difuzor
;;      Argumente: <Intrare> : -
;;                  <leire> : -
;;      Altereaza:   registrul AL
;;
PPAR     EQU    61H      ; adresa portului paralel 8255, canal B, la care se vor
                        ; programa bitii 0 si 1 pentru activare sunet la difuzor
AMASC    EQU    03H      ; masca pentru setarea bitilor 0 si 1: bitul 0 de valoare 1 va
                        ; seta semnalul Gate2 (permitand numararea numaratorului 2),
                        ; iar bitul 1 de valoare 1 va permite ieirii Out2 sa fie aplicata pe
                        ; difuzor
;ASUNET: IN     AL,     PPAR  ; citetete actuala valoare a bitilor de la portul paralel, canal
B
          OR      AL,     AMASC ; seteaza bitii 0 si 1, lasand ceilalti biti nemodificati
          OUT     PPAR,   AL    ; inscrie noua valoare la portul paralel, canal B
;
          RET          ; programare incheiata, return
=====
;;      Subrutina pentru blocarea sunetului la difuzor
;;      Argumente: <Intrare> : -
;;                  <leire> : -
;;      Altereaza:   registrul AL
;;
PPAR     EQU    61H      ; adresa portului paralel 8255, canal B, la care se vor
                        ; programa bitii 0 si 1 pentru activare sunet la difuzor
BMASC    EQU    03H      ; masca pentru resetarea bitilor 0 si 1: bitul 0 de valoare 0 va
                        ; resete semnalul Gate2 (blocand numararea numaratorului 2),
                        ; iar bitul 1 de valoare 0 nu va permite ieirii Out2 sa fie aplicata
                        ; pe difuzor
;
          Obs: poate fi resetat doar unul dintre cei doi biti, pentru blocarea sunetului

```

```

;
BSUNET: IN      AL,      PPAR      ; citește actuala valoare a bitilor de la portul paralel, canal
B
          AND     AL,      BMASC    ; resetează bitii 0 și 1, lăsând ceilalți biti nemodificați
          OUT     PPAR,    AL        ; înscrie noua valoare la portul paralel, canal B
;
          RET                                ; programare încheiată, return
;=====
;      Program principal, care apelează cele trei subrutine definite mai sus
;
;
NDATA EQU 0952H          ; valoarea calculată pentru N, pentru a genera un semnal
                          ; cu frecvență de 500 Hz la ieșirea număratorului 2 al 8253
MAIN:  MOV     BX,  NDATA    ; încarcă BX cu valoarea N
       CALL   INITIM2       ; programează număratorul 2 al 8253
       CALL   ASUNET        ; activează sunet la difuzor
;
;      ... aici, pune o temporizare, pentru a lăsa sunetul activ, pentru
;      o durată dată de timp
;
       CALL   BSUNET        ; blochează sunetul la difuzor
;
       HLT                                ; program terminat
;=====

```

Reluăm programul, prezentându-l în limbaj PASCAL, combinat cu utilizarea unora dintre secțiunile programului scrise în limbaj de asamblare. Se programează 8253, se apasă o tastă de la tastatura calculatorului pentru a activa sunetul și se blochează sunetul la următoarea apăsare a unei taste.

```
{=====}
```

Program Sunet;

const:

```

OTIMC = $43; { definește adresa portului de control al 8253 }
OTIM2 = $42; { definește adresa portului de date pentru număratorul 2 al 8253 }
PPAR  = $61; { definește adresa portului paralel, canal B, circuitul 8255 }

```

;

```

DTCMD = $B6; { cuvântul de comandă pentru 8253, 1011 0110 b, setând
              programare numărator 2, în mod 3 de numărare, încărcare

```

succesivă

```

              octet low, apoi high, cu valoare binară }
AMASC = $03; { mască pentru activarea sunetului la difuzor }
BMASC = $FC; { mască pentru blocarea sunetului la difuzor }

```

;

```

NDATA = $0952; { valoarea N pentru a obține 500 Hz la ieșirea
                număratorului 2 al 8253 }

```

```
{-----}
```

begin

asm

```

MOV     AL,      DTCMD    ; încarcă AL cu cuvântul de comandă
OUT     OTIMC,   AL        ; transmite cuvântul de comandă la 8253
MOV     AX,      NDATA    ; încarcă AX cu valoarea N
OUT     OTIM2,   AL        ; transmite low(N) la 8253, număratorul 2
OUT     OTIM2,   AH        ; transmite high(N) la 8253, număratorul 2

```

```

    end;
{-----}
    write('Apasati o tasta pentru a activa sunetul');
    readln();
{-----}
    asm
        IN      AL,      PPAR    ; citeºte actuala valoare a bitilor de la portul paralel, canal
B
        OR      AL,      AMASC   ; seteaza bitii 0 ºi 1, lasand ceilalti biti nemodificati
        OUT     PPAR,    AL      ; inscrie noua valoare la portul paralel, canal B
    end;
{-----}
    write('Apasati o tasta pentru a bloca sunetul');
    readln();
{-----}
    asm
        IN      AL,      PPAR    ; citeºte actuala valoare a bitilor de la portul paralel, canal
B
        AND     AL,      BMASC   ; reseteaza bitii 0 ºi 1, lasand ceilalti biti nemodificati
        OUT     PPAR,    AL      ; inscrie noua valoare la portul paralel, canal B
    end;
{-----}
end.
{=====}

```

Observatie: Pentru generarea de semnale audibile, spectrul de frecvente maxim ce poate fi sesizat de urechea omeneasca este in gama 15 Hz – 20 kHz. Tabela de mai jos indica valorile zecimale ce pot fi utilizate pentru a genera unele note muzicale, pe un sistem PC.

Frecventa de intrare a generatorului de semnal este de 1.193.180 Hz

\ Nota octava	Do	Re	Mi	Fa
1	18357	16345	14551	13715
2	9108	8117	7231	6818
3	4554	4058	3616	3409
4	2277	2096	1808	1705
5	1139	1015	904	852
6	569	507	452	426
7	285	254	226	213

In esenta, un salt de o octava se va obtine prin dublarea (injumatatirea) frecventei.

5.5 PORTURI SPECIALIZATE – CONTROLERE DE INTRERUPERI

Paragraful 4.5 a prezentat structura sistemului de intreruperi a microprocesorului 8086, cat și caracteristicile de baza ale controlerului de intreruperi 8259A. Figura 4.33 a prezentat pinii circuitului, iar par.4.5.3 a dat detalii despre utilizarea acestuia in sistemele cu microprocesor 8086. In continuare se vor prezenta principalele aspecte legate de programarea acestui port specializat, si unele exemple concrete de utilizare pe microcalculatoarele de tip PC.

5.5.1 Programarea controlerului de intreruperi 8259A

Dupa cum s-a precizat, tratarea intreruperilor externe mascabile este gestionata cu ajutorul circuitului programabil PIC 8259A (PIC = *Programmable Interrupt Controller*). In acest paragraf vom da detalii suplimentare, referitoare la programarea și utilizarea acestui circuit. Specificatii complete de utilizare se pot gasi in cataloagele și manualele de aplicatie oferite de firma INTEL.

Trebuie precizat ca circuitele 8259A pot fi utilizate și in montaje in cascada, avand rol de **Master** (1), respectiv **Slave** (max. 8), circuitele **Slave** avand conectate ieșirile **INT** la intrarile **IR_n** ale circuitului **Master**. Astfel, pana la 64 de semnale de intrerupere externe pot fi tratate de catre aceste structuri.

Pe scurt, pinii circuitului 8259A au urmatoarea semnificatie (v.fig. 4.33):

- D0-D7: pinii de date, permitand comunicarea cu microprocesorul, pentru programarea circuitului, testarea starii acestuia și transmiterea pe ciclul 2 INTA a codului intreruperii solicitate de 8259A.
- RD, WR: semnale de control, indicand operatia de citire, respectiv scriere in circuit;
- CS : semnalul de selectie a circuitului (Chip Select);
- A0 : bit adresa, permitand selectia unuia din cele 2 porturi interne ale circuitului (v. programarea);
- INT, INTA: semnale omonime cu cele corespunzatoare microprocesorului 8086;
- SP/EN : ca semnal de intrare, indica circuit utilizat ca Master (1), sau Slave (0); intr-un mod special de functionare a circuitului (buffer mode), este utilizat pentru a comanda circuite tampon externe.
- CAS 0-2: semnale generate in schemele cascadeate (mai multe circuite 8259A) de catre Master, catre Slave, indicand carui Slave i s-a acceptat cererea de intrerupere la un moment dat.

Circuitul 8259A este programat utilizand o secventa de pana la 4 cuvinte de initializare (**ICW** – **Initialisation Command Word**), a caror semnificatie este, pe scurt, urmatoarea (se insista pe elementele specifice utilizarii in sistemele 8086 – circuitul este utilizabil și in sisteme 8080, 8085):

ICW1:

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	X	X	X	1	LTIM	ADI	SNGL	IC4

- **IC4**: 1 = se va transmite și ICW4;

0 = nu se va transmite ICW4;

- **SNGL**: 1 = un singur circuit 8259A in sistem;

0 = mai multe circuite, cascadeate;

- **ADI:** 1 = interval de 4 adrese, 0 = interval de 8 adrese, între vectorii de intrerupere;

- **LTIM:** 1 = intrarile IRI(0-7) sunt sensibile pe nivel; 0=intrarile Iri(0-7) sunt sensibile pe front;

ICW2:

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	T7	T6	T5	T4	T3	0	0	0

- **T7-T3** : bitii vectorului de intrerupere de baza, asociat intreruperilor generate de catre circuit;

ICW3: (utilizat doar in schemele cascade):

- pentru circuitul Master:

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	S7	S6	S5	S4	S3	S2	S1	S0

Si: 1= intrarea IR asociata este conectata la un Slave; 0= intrarea IR asociata nu este conectata la un slave;

- pentru circuitul Slave:

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	0	0	0	ID2	ID1	ID0

- **ID0-2** : indica identificatorul (numarul – unic) circuitului Slave;

ICW4: utilizat pentru a specifica detalii suplimentare despre modul de lucru al circuitului (a se consulta catalogul circuitului, pentru prezentarea acestui cuvânt de initializare).

In afara cuvintelor de initializare, se utilizeaza, in functionarea normala a circuitului, a^oa-numitele cuvinte de operare (OCW – Operation Command Word), permitand modificarea unora dintre parametrii ^oi/sau modurile de lucru ale circuitului 8259A. Aceste cuvinte permit:

OCW1:

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

Mi : 1 = masca pe IRI; 0=IRi nu este mascata;

OCW2:

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	R	SL	EOI	0	0	L2	L1	L0

- valoarea uzuala a OCW2 este 20H, indicand transmiterea unui mesaj de sfarșit de intrerupere (EOI = *End Of Interrupt*), inainte de ieșirea din rutina de tratare a intreruperii. Pentru detalii suplimentare, a se consulta catalogul circuitului 8259A.

OCW4: utilizat in moduri speciale de functionare ale circuitului; a se consulta catalogul circuitului, in acest sens.

Trebuie mentionat ca, in mod implicit, cele 8 niveluri de intreruperi au prioritati fixe: maxima pentru IR0, minima pentru IR7. Se pot utiliza și moduri de lucru pentru care prioritatile se rotesc dupa fiecare tratare a unei cereri de intrerupere (nivelul tratat devine de prioritate minima, celelalte ajustandu-și prioritatea in mod corespunzator), sau chiar la indicatia specifica a utilizatorului – utilizand OCW2.

De asemenea, trebuie specificat ca, in schemele cascade, TOATE circuitele 8259A prezente in schema trebuiesc programate in mod individual. Trebuie indicate (hardware sau software) circuitul Master și circuitele Slave; pentru fiecare in parte se transmit cuvintele de initializare (ICW), indicandu-se adresele vectorilor de intrerupere asociati (DISTINCTE !) etc. La aparitia unei cereri de intrerupere la o intrare IR a unui circuit Slave, acesta va genera o cerere de intrerupere catre circuitul Master, pe intrarea IR corespunzatoare, la care este conectata ieșirea INT a circuitului Slave respectiv. In ciclul INTA, circuitul Master va indica, pe liniile CAS0-2, identificatorul circuitului Slave a carui cerere de intrerupere a fost transmisa microprocesorului și acceptata. Apoi, circuitul Slave va fi cel ce furnizeaza pe magistrala de date a sistemului codul intreruperii solicitate, permitand localizarea vectorului de intrerupere și a rutinei de tratare a intreruperii, asociate acestei intreruperi. Pentru fiecare dintre circuitele din schema se aplica logica de prioritate programata pentru circuitul respectiv, permitand implementarea unor scheme sofisticate de rezolvarea a prioritatilor in cazul cererilor de intrerupere simultane. De mentionat ca este posibil ca doar unele dintre intrarile IR ale circuitului Master sa fie conectate la circuite Slave, celelalte fiind conectate direct la echipamente de I/E (v. par. urmator).

5.5.2 Tratarea intreruperilor externe mascabile in microcalculatoarele PC-AT

Microcalculatoarele PC-AT utilizeaza 2 circuite 8259A in conexiune Master-Slave, cu urmatoarea configuratie:

Tabel 5.3. Asignarea nivelurilor de intreruperi in microcalculatoarele PC-AT.

CTRL1 (Master)	CTRL2 (Slave)	Nivel intrerupere	Utilizare
IRQ0		8h	Ieșire 0 a circuit ceas 8254
IRQ1		9h	Tastatura (buffer ieșire plin)
IRQ2 <---	IRQ8	70h	Intrerupere ceas timp real al sistemului
	IRQ9	71h	Redirectata software la INT 0Ah (IRQ 2)
	IRQ10	72h	Rezervat
	IRQ11	73h	Rezervat
	IRQ12	74h	Rezervat

	IRQ13	75h	Coprocessor
	IRQ14	76h	Controler hard disk
	IRQ15	77h	Rezervat
IRQ3		0Ah	Port serial COM2
IRQ4		0Bh	Port serial COM1
IRQ5		0Ch	Port paralel 2
IRQ6		0Dh	Controler disc flexibil
IRQ7		0Eh	Port paralel 1

5.5.3 Exemple de programe de tratarea intreruperilor

In exemplele urmatoare, se vor implementa programe tratand intreruperea de la circuitul de ceas 8254 (ie^{ire} 0). Pentru detalii privind programarea ^oi utilizarea acestui circuit, a se consulta paragraful 5.4.

Observatie: Se utilizeaza, in exemplele urmatoare, functii specifice pentru a se genera/salva vectorii de intreruperi (**SetIntVect**, **GetIntVect**) ^oi a genera o procedura specifica tratarii unei intreruperi, conforma cu caracteristicile limbajului de programare Pascal. A se consulta manualul acestuia sau facilitatea “**Help on-line**” pentru detalii referitoare la aceste functii.

0. Intreruperi software de tip general

```
{-----}
{ Test 0: intreruperi software }
{ Programul genereaza intreruperea 35H, unde alternativ se
  activeaza / oprete generarea unui sunet la difuzor }

uses Dos;
const
  NDATA = $0952; { val.pt. a obtine 500 Hz la ieire numaratorului 2 al 8253 }
  start : Boolean = true;
var
  Int35Save : Pointer;
  regs : Registers;
{-----}
{ Initializare ceas de timp real (NUM.2, 500 Hz la ieire) }
procedure InitCeas;
begin
  asm
    MOV     AL,    $B6    { incarca AL cu cuvantul de comanda }
    OUT$43, AL        { transmite cuvantul de comanda la 8253 }
    MOV     AX,    NDATA { incarca AX cu valoarea N }
    OUT$42, AL        { transmite low(N) la 8253, numaratorului 2 }
    MOV     AL,    AH    { transfera high(N) in AL }
    OUT$42, AL        { transmite high(N) la 8253, numaratorului 2 }
  end;
end;
{-----}
procedure SunetOn;
```

```
begin
asm
IN  AL,    $61    { cite°te bitii de la portul paralel, canal B }
OR  AL,    $03    { seteaza bitii 0 °i 1, lasand ceilalti biti nemodificati }
OUT $61,   AL     { inscrie noua valoare la portul paralel, canal B }
end;
end;
{-----}
{ Stop sunet la difuzor }
procedure SunetOff;
begin
asm
IN  AL,    $61    { cite°te bitii de la portul paralel, canal B }
AND AL,    $FC    { reseteaza bitii 0 °i 1, lasind ceilalti biti nemodificati }
OUT $61,   AL     { inscrie noua valoare la portul paralel, canal B }
end;
end;
{-----}
{$F+}
{ Procedura de intrerupere de nivel 35H (RTI 35H) }
procedure Intrerupere35; interrupt;
begin
if(start) then SunetOn else SunetOff;
WriteLn('Intrerupere 35H! start =', start);
ReadLn;
start := FALSE;
end;
{$F-}

begin
{-----}
{ Program principal }
{-----}
{ Initializare vector de intreruperi pt. nivelul 35 }
GetIntVec($35,Int35Save); { Salvare vechiul vector de intreruperi }
SetIntVec($35,Addr(Intrerupere35)); { Setare noul vector de intreruperi }
{-----}
InitCeas; { Initializare ceas 2 al 8253 }
{-----}
{ >>>> }
WriteLn('Apasati o tasta pentru a genera intrerupere 35');
ReadLn;
Intr($35, regs);
WriteLn('Apasati o tasta pentru a genera intrerupere 35');
ReadLn;
Intr($35, regs);
{<<<<}
{-----}
SetIntVec($35,Int35Save); { Refacere vector de intrerupere de nivel 35 }
end.
```

1. Intreruperea interna de impartire prin 0.

În acest caz, se pot efectua modificări minore în exemplul de mai sus, legate în esență de următoarele aspecte:

- întreruperea de împărțire prin 0 generează întrerupere pe nivel 0 al microprocesorului (această întrerupere este generată și dacă, în urma unei împărțiri, rezultatul acesteia nu poate fi reprezentat pe precizia cu care s-a efectuat operația). Astfel, secvențe de genul:

```

MOV AX, 2000
MOV BL, 2
DIV BL
sau
MOV AX, 10
MOV BL, 0
DIV BL

```

vor genera această întrerupere. În programul de mai sus, se pot efectua următoarele modificări:

- se elimină toate secvențele referitoare la utilizarea circuitului de ceas;
- se modifică nivelul de întrerupere de la 35h, la 0h (peste tot unde este cazul);
- în rutina de tratare a întreruperii, se poate da un mesaj specific și seta un indicator de eroare;
- se înlocuiește secvența de instrucțiuni dintre liniile {>>>>} și {<<<<}, cu o buclă în care, de exemplu, se citesc de la tastatură 2 numere (unul de tip integer, celălalt shortint), care apoi se împart; rezultatul (shortint) se afișează. Secvența de calcul se poate scrie în assembler. Utilizând cazuri care generează întrerupere de nivel 0, se poate verifica programul.
- dacă nu apare eroare (se testează indicatorul de eroare), se tipărește rezultatul; altfel, un mesaj de eroare.

```

{-----}
{ Test 1: intreruperi interne }
{ Programul trateaza intreruperea 0 }

uses Dos;

const
  Eroare : Boolean = FALSE;
  N1: Integer = 0;

var
  Int0Save : Pointer;
  N2: Shortint;
  N3: Shortint;
{-----}
{$F+}
{ Procedura de intrerupere de nivel 0H }
procedure Intrerupere0; interrupt;
begin
  Write('Intrerupere 0H! Eroare la impartire !');
  ReadLn;
  Eroare := TRUE;

```

```
end;
{$F-}

begin
{-----}
{ Program principal }
{-----}
{ Initializare vector de intreruperi pt. nivelul 0 }
  GetIntVec($0,Int0Save); { Salvare vechiul vector de intreruperi }
  SetIntVec($0,Addr(Intrerupere0)); { Setare noul vector de intreruperi }
{-----}
  while (n1 <> -1) do
  begin
    Write('Introduceti n1:');
    ReadLn(n1);
    Write('Introduceti n2:');
    ReadLn(n2);

    asm
      MOV AX, N1
      MOV BL, N2
      DIV BL
      MOV N3, AL
    end;

    if Eroare then
    begin
      WriteLn('Rezultat = INFINIT');
      Eroare := FALSE;
    end
    else WriteLn('Rezultat =',n3);
  end;
{-----}
  SetIntVec($0,Int0Save); { Refacere vector de intrerupere de nivel 0 }

end.
```

2. Intreruperea interna de depa^oire (overflow).

Se trateaza similar cazului de mai sus, cu urmatoarele mentiuni suplimentare:

- nivelul intreruperii este 4;
- intreruperea este generata la aparitia unei depa^oiri de calcul ^oi testata automat intr-o secventa de genul:

```
MOV BL, N2
ADD AL, BL
INTO ; daca a aparut overflow, va genera INT 4
MOV N3, AL
```

Se prezinta in continuare programul:

```
{-----}
```

```

{ Test 1: intreruperi interne }
{ Programul trateaza intreruperea 4 - Overflow }

uses Dos;

const
  Eroare : Boolean = FALSE;
  N1: ShortInt = 0;

var
  Int4Save : Pointer;
  N2: ShortInt;
  N3: ShortInt;
{-----}
{$F+}
{ Procedura de intrerupere de nivel 4H }
procedure Intrerupere4; interrupt;
begin
  Write('Intrerupere 4H! Depasire !');
  ReadLn;
  Eroare := TRUE;
end;
{$F-}

begin
{-----}
{ Program principal }
{-----}
{ Initializare vector de intreruperi pt. nivelul 4 }
GetIntVec($4,Int4Save); { Salvare vechiul vector de intreruperi }
SetIntVec($4,Addr(Intrerupere4)); { Setare noul vector de intreruperi }
{-----}
while (n1 <> -1) do
begin
  Write('Introduceti n1:');
  ReadLn(n1);
  Write('Introduceti n2:');
  ReadLn(n2);

  asm
    MOV AL, N1
    MOV BL, N2
    ADD AL, BL
    INTO
    MOV N3, AL
  end;

  if Eroare then
  begin
    WriteLn('Rezultat = DEPASIRE');
    Eroare := FALSE;
  end
  else WriteLn('Rezultat =',n3);
end;
{-----}

```

```
SetIntVec($4,Int4Save); { Refacere vector de intrerupere de nivel 4 }
```

```
end.
```

3. Intreruperea externa de la circuitul de ceas 8254, contor

Se utilizeaza informatiile prezentate in lucrarea prezenta și in lucrarea “*Porturi specializate. Ceasuri programabile. Intrefete seriale*”. Deoarece circuitele 8259A ale microcalculatorului PC sunt deja programate, este necesar sa se utilizeze doar cuvintele de operare: OCW1 pentru a se modifica masca intreruperii de nivel 0 a circuitului Master (v. Tabel 2) și OCW2 pentru a se transmite comanda de sfârșit de intrerupere catre circuitul 8259A, la sfârșitul rutinei de tratare a intreruperii respective. In aceste conditii, un posibil exemplu de program care utilizeaza aceasta intrerupere este urmatorul (programul permite citirea de caractere de la consola, iar in rutina de intrerupere, deoarece se poate programa o rata a intreruperilor de minim 18,7 Hz, s-a programat o rata – mai exacta – de 25 Hz și, utilizand un contor software, la 25 de intreruperi – o data pe secunda – se tiparește un alt caracter, ‘X’) :

```
{-----}
{ Test 3: intreruperi hardware }
{ Programul trateaza intreruperea de nivel 8.
  Aceasta provine de la PIC0, IR0,
  generata la aparitia unui semnal de la 8253, canal 0 }

uses Dos, Crt;
label
  bucla;
const
  NDATA = 46800; { val.pt. a obtine 25 Hz la ieșirea numaratorului 0 al 8253 }
  contor : Integer = 0;
  PIC_OCW1 = $20; { adresa 8259, OCW1 }
  PIC_OCW2 = $21; { adresa 8259, OCW2 }
  ch: char = 'a';

var
  Int8Save : Pointer;
  regs : Registers;
  Mascve: ShortInt;
{-----}
{ Initializare ceas de timp real (NUM.0, 25 Hz la ieșire) }
procedure InitCeaș0;
begin
  asm
    MOV     AL,    $36    { incarca AL cu cuvantul de comanda }
    OUT$43, AL    { transmite cuvantul de comanda la 8253 }
    MOV     AX,    NDATA { incarca AX cu valoarea N }
    OUT$40, AL    { transmite low(N) la 8253, numaratorul 0 }
    MOV     AL, AH   { transfera high(N) in AL }
    OUT$40, AL    { transmite high(N) la 8253, numaratorul 0 }
  end;
end;

{-----}
```

```

{$F+}
{ Procedura de intrerupere de nivel 8H (RTI 8H) }
{ Procedura este lansata de catre semnalul de la 8253, la fiecare 40 ms }
procedure Intrerupere8; interrupt;
begin
  contor:=contor+1;
  if (contor > 25) then
  begin
    { odata pe secunda tipare°te 'X' }
    write('X');
    contor := 0;
  end;

  asm
    MOV DX, $20 { adresa PIC-OCW1, pt. a da EOI }
    MOV AL, $20 { in AL, cod EOI }
    OUT DX, AL { se da EOI la PIC }
  end;
end;
{$F-}

begin
  {-----}
  { Program principal }
  {-----}
  { Initializare vector de intreruperi pt. nivelul 8 }
  GetIntVec($8,Int8Save); { Salvare vechiul vector de intreruperi }
  SetIntVec($8,Addr(Intrerupere8)); { Setare noul vector de intreruperi }
  {-----}
  InitCea0; { Initializare °i start ceas 0 al 8253 }
  {-----}
  { Initializare PIC }
  asm
    MOV DX, $21 { in DX, adresa OCW2 – Mask port }
    IN AL, DX { citire masca intreruperi curente }
    MOV MASCVe, AL { memorare masca veche }
    AND AL, $FE { demascare bit 0, pt. IR0 }
    OUT DX, AL { programare noua masca }
    STI { activare intreruperi }
  end;
  {-----}
  { In aceasta bucla se citesc caractere de la tastatura }
  { La cate o secunda, rutina de intrerupere tipare°te caracterul "X" }
  while (ch <> ' ') do { Se iese din program apasand "Spatiu" }
  begin
    if Keypressed then
    begin
      read(ch);
      write(ch);
    end;
  end;
  {-----}
  { refacere masca initiala PIC }
  asm
    MOV DX, $21 { in DX, adresa OCW2 – Mask port }

```

```
    MOV AL, MASCVE { memorare masca veche }  
    OUT DX, AL { programare noua masca }  
end;  
{-----}  
SetIntVec($8,Int8Save); { Refacere vector de intrerupere de nivel 8 }  
end.
```

6 APLICATII ALE SISTEMELOR MICROPROCESOR

Capitolele precedente au incercat sa creeze o imagine cat mai precisa in ceea ce priveste elementele componente definitorii ale sistemelor microprocesor, prezentarea caracteristicilor lor functionale, a modului de implementare a structurilor hardware de principiu si a metodelor de utilizare prin program a acestor resurse ale sistemelor respective. Prezentarea facuta in capitolul 3 a avut ca principal scop familiarizarea cititorului cu toate aspectele de principiu privind sistemele microprocesor. Capitolul 4 a concretizat studiul microprocesoarelor cu unul dintre tipurile cele mai reprezentative pentru acestea — circuitul INTEL 8086 — oferind detalii concrete referitoare la parametrii, arhitectura interna si externa a microsistemelor realizate cu acest tip de circuit si a circuitelor aferente utilizate la constructia sistemelor respective. In fine, capitolul 5 a prezentat principalele tipuri de porturi specializate si interfete utilizate pentru conectarea diferitelor periferice uzuale folosite in sistemele microprocesor.

Se poate astfel considera ca, in acest stadiu al lucrarii, pe baza elementelor generale prezentate, cititorul este in masura sa abordeze problematica realizarii in detaliu a unui asemenea sistem, in vederea implementarii unor aplicatii industriale ce utilizeaza sistemul microprocesor ca element central de comanda si control. Doar printr-o exemplificare concreta, detaliata, a unor aplicatii reprezentative, se vor putea evidentia aspecte practice a caror prezentare a fost uneori in mod voit evitata in prezentarea din capitolele precedente, cu scopul de a oferi o abordare treptata, cu un grad din ce in ce mai mare de dificultate, a problematicii conceperii, proiectarii si implementarii practice a acestor sisteme. Numeroase “mici” detalii, a caror prezentare ar fi ingreunat urmarirea elementelor fundamentale ce erau descrise, vor putea fi astfel evidentiata, cu scopul de a oferi cititorului nu doar un material de principiu, ci si un punct de pornire pentru eventualii proiectanti ai acestor microcalculatoare. Totodata, chiar si pentru categoria, probabil, mult mai larga, a viitorilor utilizatori ai unor structuri hardware existente, pentru a implementa prin software aplicatii specifice exemplele urmatoare vor constitui elemente foarte utile. Si aceasta, reamintim din nou, deoarece in cazul aplicatiilor industriale ale sistemelor microprocesor legatura hardware-software este foarte stransa, iar o aplicatie performanta nu se poate implementa decat prin cunoasterea temeinica a ambelor laturi definitorii ale acestor sisteme.

Exemplul ce va fi prezentat in continuare a fost ales din gama, practic nelimitata, a aplicatiilor ce pot fi concepute cu un sistem microprocesor, datorita unor caracteristici ce au fost considerate reprezentative pentru scopul urmarit:

- aplicatia aleasa reprezinta o aplicatie de complexitate ridicata, justificand necesitatea utilizarii unui sistem microprocesor performant pentru implementarea sa;
- structura hardware necesara acestei aplicatii contine, practic, toate tipurile de componente reprezentative (memorie, porturi), utilizate in mai mica sau mai mare masura in majoritatea aplicatiilor de proces ale sistemelor microprocesor;
- de asemenea, structura software necesar a fi implementata, de o complexitate relativ ridicata, este reprezentativa pentru gama de componente definitorii ale programelor aplicatiilor de acest fel;
- metodologia conceperii structurii sistemului, a configurarii hardware si software a acestuia, sunt valabile pentru marea majoritate a aplicatiilor industriale ale sistemelor microprocesor.

Sistem pentru controlul numeric al pozitiei unei actionari electrice, realizat cu microprocesorul 8086

Consideram urmatoarea tema de proiectare: Sa se implementeze un sistem de control discret al unei actionari electrice cu motor de c.c.

Problema controlului discret al sistemelor de actionare electrica reprezinta o aplicatie de varf pentru realizatorii acestor sisteme, datorita performantelor extrem de ridicate pe care trebuie sa le aiba componentele utilizate: motoare cu caracteristici dinamice foarte ridicate (constante de timp mecanice de ordinul milisecundelor), convertizoare de putere realizate uzual cu tranzistoare (MOSFET sau bipolare), functionand la frecvente ridicate (zeci de kHz), senzori si traductoare aferente si, in fine, sisteme de control suficient de puternice, cu facilitati de calcul (viteza, precizie, set de instructiuni) si resurse (memorie, interfete) care sa permita controlul in timp real al acestor componente electromecanice.

Din start trebuie precizat ca, pentru o asemenea aplicatie, pentru un sistem de actionare rapid, va fi necesar un sistem de control realizat cu un microprocesor de 16 biti, de viteza suficient de ridicata pentru a se putea implementa structura de control cu constante de timp de esantionare de ordinul milisecundelor, sau chiar mai reduse. Microprocesorul 8086 poate reprezenta o alternativa pentru implementarea acestor sisteme, alte solutii ce ar trebui considerate fiind reprezentate de utilizarea unui microprocesor de prelucrare a semnalelor, DSP, sau a unui microcontroler pe un cip, de 16 biti.

Deoarece implementarea inclusiv a buclei de curent in sistemul numeric nu poate fi asigurata intr-un sistem cu microprocesor de tip 8086 (perioada de esantionare pentru aceasta bucla poate ajunge pentru unele motoare chiar la ordinul zecilor de microsecunde), se va presupune ca aceasta bucla este realizata in sistemul analogic, clasic. De altfel, aceasta abordare este mult mai realista din punct de vedere al costului implicat (ar putea fi utilizat un microprocesor mult mai rapid, dar la un pret mult sporit), cat si al faptului ca o bucla analogica de curent se poate implementa relativ simplu si nu necesita o structura complexa a regulatorului, uzual o schema de tip PI fiind suficienta in acest sens.

Astfel, sistemul numeric va putea implementa atat bucla de control al vitezei cat si cea de control al pozitiei, cu avantajele cunoscute ale flexibilitatii, ale posibilitatii implementarii de structuri adaptive, de proceduri de identificare a parametrilor sistemului si de auto-acordare a schemei de reglaj etc.

Plecand de la enuntarea temei propuse, dupa cum s-a precizat si in capitolul 1 al acestei lucrari, va trebui urmata o anumita metodologie pentru a se putea realiza o proiectare eficienta si corecta a unui asemenea sistem. Principalele etape pe care le vom parcurge in paragrafele urmatoare, pentru a configura sistemul de control dorit, vor fi:

- (a) evaluarea performantelor impuse sistemului si, de aici, alegerea configuratiei acestuia;
- (b) proiectarea schemei de control numeric;
- (c) proiectarea structurii hardware a sistemului;
- (d) proiectarea structurii software a sistemului;
- (e) concluzii.

Prezentarea se va axa, in conformitate cu scopul acestei lucrari, cu prioritate pe evidentiarea aspectelor legate in mod direct de proiectarea sistemului microprocesor. Doar in masura in care e necesara precizarea calitativa a unor aspecte legate de proiectarea sistemului in ansamblu, se va face referire si la elemente definitorii legate de modelarea sistemului, la aspectele teoretice ale proiectarii schemei de control numeric etc. Aceste elemente sunt, de altfel, aprofundate in detaliu in lucrari dedicate studiului servomecanismelor, sau al teoriei sistemelor si a reglajului automat (v. bibliografia).

6.1 PERFORMANTELE IMPUSE SISTEMULUI DE CONTROL NUMERIC. ALEGEREA CONFIGURATIEI SISTEMULUI.

Pornind de la performantele impuse de catre utilizator pentru aplicatia ce se va implementa, se va putea decide tipul componentelor si evalua caracteristicile acestora pentru sistemul de control numeric propus. In esenta, utilizatorul va impune, pornind de la specificatiile sistemului existent pe de o parte, si de la specificatiile sistemului final, controlat, pe de alta parte, doua categorii de caracteristici ce vor sta la baza configurarii intregului sistem:

- **caracteristici globale initiale** (tipul si parametrii sarcinii, tipul si parametrii traiectoriilor de miscare etc.);
- **caracteristicile globale finale** (performante si parametri ai sistemului de reglare: comportare dinamica, precizie, robustete etc.).

Pe baza specificarii acestor elemente definitorii, se va putea defini tipul si specifica parametrii de baza ai diverselor componente din sistem, ajungand, in final, la alegerea corecta a acestora.

Figura 6.1 prezinta o posibila schema de principiu a structurii decizionale continand principalele aspecte ce trebuiesc studiate, in vederea unei abordari corecte si complete a acestei problematice.

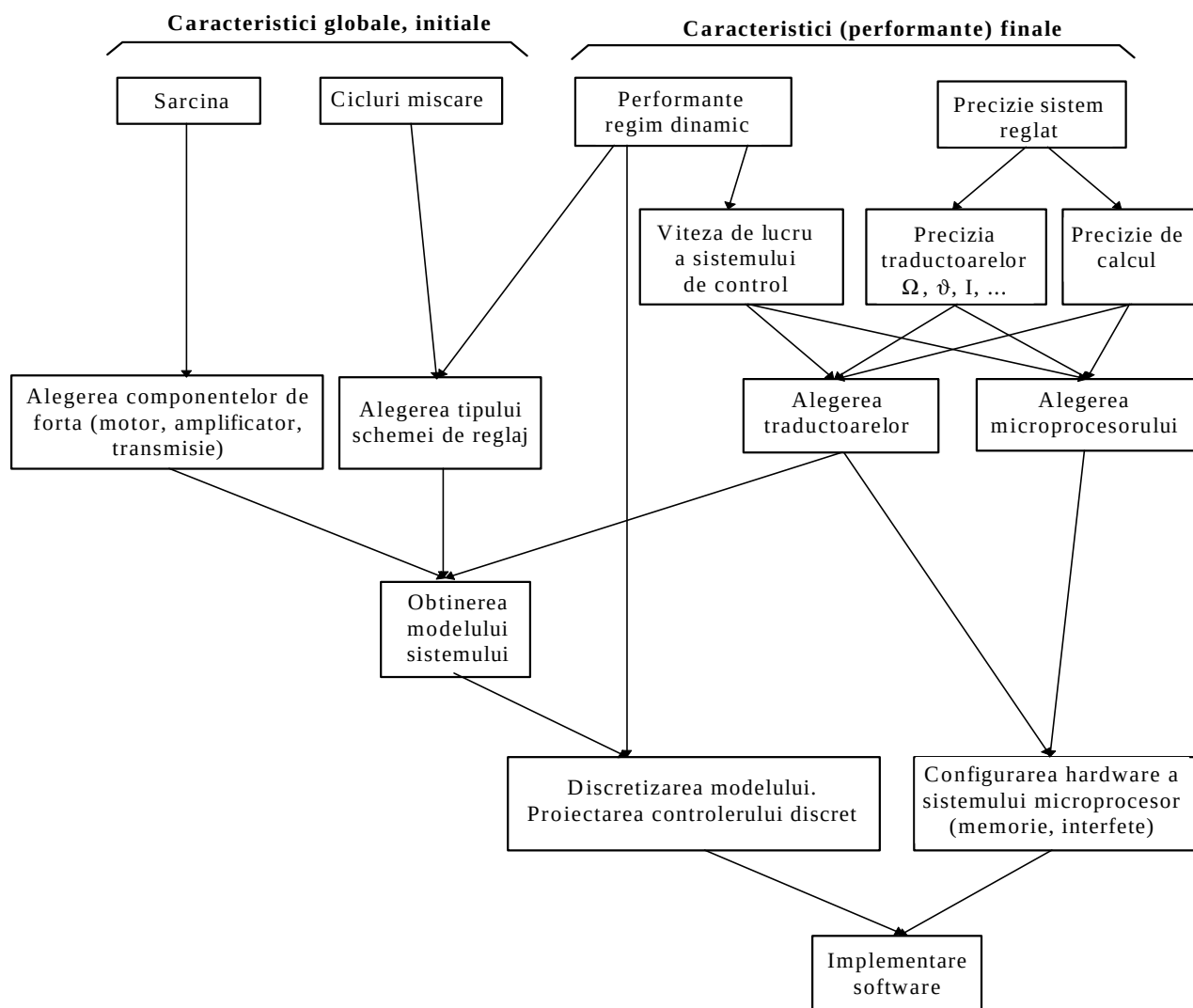


Figura 6.1. Diagrama decizională pentru configurarea și proiectarea sistemului de control al unei acționări electrice

Dupa cum se poate remarca din figura, se includ in categoria *caracteristicilor globale, initiale* ale sistemului, parametrii și specificatiile ce definesc *sarcina* și *ciclurile de miscare* ce vor trebui sa fie efectuate de catre sistem.

6.1.1 Sarcina

Sarcina este impusa de aplicatia propriu-zisa, iar utilizatorul trebuie sa aiba specificati toti parametrii de baza, definitorii, ai acesteia:

- *tipul de miscare a sarcinii*: rotativa sau liniara;
- *parametrii statici de baza* :
 - *momentul de inertie* total al sarcinii, J_s ;

- cuplul de frecari statice, M_s (pentru sarcini rotative), respectiv forta de frecari statice, F_{fs} (pentru sarcini liniare);
- coeficientul de frecari vascoase, F_s ;
- componentele dinamice ale cuplului (ale fortei - in cazul miscarii liniare) de sarcina:
 - componenta dependenta de timp: $M_s(t)$ in cazul miscarii rotative, respectiv $F_s(t)$ in cazul miscarii liniare;
 - componenta dependenta de viteza: $M_s(\Omega)$ in cazul miscarii rotative, respectiv $F_s(v)$ in cazul miscarii liniare;
 - componenta dependenta de pozitie: $M_s(\vartheta)$ in cazul miscarii rotative, respectiv $F_s(x)$ in cazul miscarii liniare
- parametrii limita de miscare:
 - acceleratia maxima a sarcinii: ε_{smax} in cazul miscarii rotative, respectiv a_{smax} in cazul miscarii liniare;
 - viteza maxima a sarcinii: Ω_{smax} in cazul miscarii rotative, respectiv v_{smax} in cazul miscarii liniare;
 - viteza minima a sarcinii: Ω_{smin} in cazul miscarii rotative, respectiv v_{smin} in cazul miscarii liniare

6.1.2 Ciclul de miscare

Ciclul de miscare este impus de asemenea de catre aplicatia ce se va implementa. Legat de ciclul de miscare, trebuiesc precizate tipul serviciului de functionare al sarcinii, eventual forma semnalului de referinta de pozitie, cat si durata acestuia. Eventuala utilizare a profilelor de viteza va trebui, de asemenea, sa fie specificata.

Dupa cum se poate remarca din figura 6.1, se includ in categoria caracteristicilor globale finale ale sistemului parametri si specificatiile ce definesc **performantele de regim dinamic** si **precizia sistemului reglat** ce vor trebui sa fie asigurate de catre sistem.

6.1.3 Performantele de regim dinamic

Aceste performante vor fi impuse de catre utilizator, fiind obtinute pe baza unor considerente legate de caracteristicile si cerintele sistemului reglat. Ele sunt uzual impuse, pentru un anumit tip de semnal de referinta, de genul treapta, rampa sau sinusoidal. Astfel, se vor specifica urmatoorii parametri:

- timpul tranzitoriu $t_{tranzit}$, al raspunsului sistemului, sau pulsatia proprie a sistemului reglat, ω_n ;
- suprareglajul raspunsului sistemului, σ , sau factorul de amortizare, ξ ;

6.1.4 Precizia sistemului reglat

Precizia sistemului reglat este, de asemenea, o caracteristica importanta, impusa de catre utilizator, si reprezinta unul dintre parametrii de performanta de baza ce trebuie asigurati de sistemul de control.

Principalii parametri ce trebuiesc specificati in aceasta categorie vor fi:

- gama de variatie a marimii de iesire (pozitie): $[\Theta_{\min}, \Theta_{\max}]$ in cazul miscarii rotative, respectiv $[x_{\min}, x_{\max}]$ in cazul miscarii liniare;
- precizia cu care trebuie efectuata pozitionarea sarcinii: $d\Theta_{\min}$ in cazul miscarii rotative, respectiv $dx_{\min}$ in cazul miscarii liniare;

Pe baza acestor parametri se poate determina **rezolutia** sistemului reglat, deci acuratetea cu care trebuie sa fie efectuata reglarea.

6.1.5 Viteza de lucru a sistemului de control

Aceasta caracteristica va rezulta in mod direct din **performantele de regim dinamic** impuse sistemului. Acest parametru va fi direct corelat, deci va impune la randul sau perioada de esantionare a buclei de reglare discrete si, indirect, va conditiona si frecventa de lucru a microprocesorului. Uzual, pentru sistemele de pozitionare care sunt implementate cu control discret al pozitiei (si al vitezei) si bucle analogice de control al curentului masinii electrice, vor fi necesare perioade de esantionare de ordinul milisecundelor sau, eventual, inferioare (sute de microsecunde) in cazul sistemelor cu constante de timp mecanice foarte reduse.

6.1.6 Precizia traductoarelor

Precizia traductoarelor din sistem va fi impusa, implicit, de **precizia sistemului reglat**. Astfel, precizia traductorului de pozitie pentru o miscare rotativa va fi impusa de **precizia de pozitionare**, $d\Theta_{\min}$. In ceea ce priveste **rezolutia** interfetei de conversie analog-numerica a pozitiei, aceasta se va calcula pe baza raportului dintre valoarea maxima a plajei de variatie a pozitiei ($\Theta_{\max} - \Theta_{\min}$) si **precizia de pozitionare**, $d\Theta_{\min}$, rotunjit superior la cea mai apropiata putere a lui 2. Astfel, plaja de variatie a pozitiei convertite va fi (in biti) data de relatia:

$$\text{plaja}_{\text{biti\_tp}} = (\Theta_{\max} - \Theta_{\min}) / d\Theta_{\min} \quad (6.1)$$

iar **rezolutia traductorului de pozitie** va fi data de valoarea $n_{\text{biti\_tp}}$, care satisface relatia:

$$2^{n_{\text{biti\_tp}}-1} < \text{plaja}_{\text{biti\_tp}} \leq 2^{n_{\text{biti\_tp}}} \quad (6.2)$$

Astfel, de exemplu, pentru o plaja de variatie a pozitiei de 360° si o precizie a pozitionarii de 30° , se obtine $\text{plaja}_{\text{biti\_tp}} = 360/30 = 12 < 16 = 2^4$, deci $n_{\text{biti\_tp}} = 4$ biti; pentru o precizie a pozitionarii de 1° , $\text{plaja}_{\text{biti\_tp}} = 360/1 = 360 < 512 = 2^9$, iar $n_{\text{biti\_tp}} = 9$ biti.

6.1.7 Precizia de calcul

Precizia de calcul va fi de asemenea impusa implicit de **precizia sistemului reglat**. Astfel, se poate evalua:

- acuratetea reprezentarii matematice utilizate (numarul de biti utilizat pentru memorarea variabilelor si pentru efectuarea calculelor matematice);
- tipul reprezentarii matematice utilizate: **numere intregi**, in virgula **fixa**, sau in virgula **mobila**.

Uzual, in cazul utilizarii numerelor *intregi* sau a celor in virgula *fixa*, acuratetea reprezentarilor numerice este de **16** biti. In cazul utilizarii numerelor in virgula *mobila*, *acuratetea* reprezentarilor numerice este de **32** de biti (**24** pentru *mantisa* si **8** pentru *exponent*). Daca reprezentarea in numere *intregi* sau in virgula *fixa* are avantajul efectuarii operatiilor mai rapid, posibil cu microprocesoare de performante mai putin elevate, principala problema in acest caz, pentru implementarea algoritmilor de control, este data de necesitatea **scalarilor** si a **testarilor depasirilor** ce pot aparea in cursul functionarii sistemului de control numeric. Aceste operatii de scalare si tratare a depasirilor nu sunt deloc usor de implementat si sunt controlat, necesitand o analiza atenta si, uneori, simulari intensive ale functionarii sistemului, pentru a asigura o functionare corecta a acestuia. Pe de alta parte, calculul in virgula *mobila* elimina toate aceste inconveniente—cu pretul, deloc redus, al utilizarii unui microprocesor de performante mult sporite (sau a unui co-procesor aritmetic, de tipul **8087**)—ceea ce conduce, evident, la cresterea costului sistemului, uneori peste limitele admisibile. Decizia solutiei adoptate va depinde de specificatiile concrete ale aplicatiei respective.

6.1.8 Alegerea componentelor de forta

Alegerea componentelor de forta reprezinta o etapa ce poate fi solutionata pe baza datelor furnizate din definirea **sarcinii**, respectiv a **ciclului de miscare** a acesteia. Principalele componente ce vor putea fi definite in acest stadiu al proiectarii vor fi: **motorul**, **amplificatorul** si **elementele de transmisie** din sistem.

- **Motorul** va fi dimensionat pe criterii energetice si din parametrii limita specificati ai sarcinii si ai miscarii. Astfel, vor fi considerate *puterea maxima* solicitata motorului in cursul miscarii, *cuplul si turatia maxime*, atat in regim permanent cat si in regim de scurta durata. Eventuale *calcule termice* vor completa aceste evaluari. Pe aceste baze se va putea alege dintr-un catalog de produse, sau se va putea elabora o tema de proiectare continand caracteristicile ce trebuiesc a fi satisfacute de motor.

Pentru tema abordata, se considera ca se utilizeaza un servomotor de c.c. cu magneti permanenti, ai carui parametri nominali de catalog vor fi cunoscuti, necesari la elaborarea modelului sistemului si la proiectarea schemei de control discret a acestuia. Astfel, principalii parametri necesari in acest caz vor fi: $n_{\max}$ (turatia maxima), R_A (rezistenta rotorica), L_A (inductivitatea rotorica), J_M (momentul de inertie al motorului), T_A (constanta de timp electrica), T_{em} (constanta de timp electromecanica), $I_{\max}$ (curentul permanent maxim), I_{imp} (curentul impulsional), K_T (constanta de cuplu).

In sistemele de actionare moderne, uzual servomotorul se realizeaza intr-o structura modulara cu **tahogenerator** si **frana electromagnetica** incorporate. Considerand ca acesta este si cazul exemplului ales, va mai fi definita odata cu motorul, ca parametru necesar, si constanta tahogeneratorului, k_{TG} . De asemenea, pentru frana electromagnetica va fi necesara comanda acesteia, cu un semnal de tip binar (0/1, inchis/deschis), deci cu un bit al unui port de iesire numerica al sistemului microprocesor. Cunoasterea parametrilor franei, si anume tensiunea de alimentare U_f , respectiv curentul absorbit de frana, I_f , vor permite dimensionarea elementului de comanda al acesteia (etajul de alimentare al franei, uzual realizat cu un tranzistor de putere cu caracteristicile compatibile cu acesti parametri).

- **Amplificatorul de putere**, care va alimenta motorul, trebuie dimensionat la o putere si la un curent furnizate cel putin egale cu cele solicitate de motor. Necesitatea de a implementa o schema reversibila va impune folosirea unui convertizor, uzual implementat cu tranzistoare, cu functionare in 4 cadrane. Parametrii definitorii, in acest caz, vor fi valorile maxime ale curentului in regim permanent, $I_{c\max}$, tensiunii, $U_{c\max}$, si curentului impulsional I_{camp} pe care convertizorul le poate furniza.

De asemenea, un parametru de interes va fi plaja tensiunii de comanda, U_c , care va fi furnizata de catre o iesire analogica (obtinuta prin utilizarea unui convertor D/A) a sistemului microprocesor. Uzual, plaja de valori pentru U_c este de $[-10, +10]$ V sau de $[-5, +5]$ V, compatibila cu iesirea tipurilor standard de convertoare D/A. Rezolutia convertorului D/A utilizat va depinde de sensibilitatea amplificatorului. Uzual, o rezolutie de 10 sau 12 biti va fi necesara pentru a asigura o functionare corecta a sistemului. Presupunem ca, pentru aplicatia propusa, s-a utilizat un convertor de 10 biti, ceea ce asigura o precizie de sub 0.1 % pentru marimea de comanda a amplificatorului.

Uzual, amplificatoarele moderne sunt dotate cu anumite facilitati de diagnoza si comanda, care vor putea fi conectate la sistemul digital, oferind avantajul de a se putea semnaliza in mod automat, la aparitia unei avarii a amplificatorului, situatia existenta si localizarea defectului (diagnoza), respectiv posibilitatea de a se comanda intr-o secventa automata functionarea acestuia (comenzi).

Astfel, **semnalizarile** utilizabile pentru implementarea unor functii de **diagnoza** automata a starii amplificatorului vor putea fi, in cazul exemplului abordat:

- indicator de functionare corecta a amplificatorului (OK general);
- indicatori de avarie:
 - viteza limita depasita ($\Omega > \Omega_{\max}$);

- curent limita depasit ($i > i_{\max}$);
- temperatura limita depasita ($t > t_{\max}$);
- atingere limita cursa stanga de miscare;
- atingere limita cursa dreapta de miscare.

Aceste semnalizari vor fi implementate ca biti ai unui port de intrari numerice, fiecare cu semnificatia indicata de numele semnalului, cu o conventie prestabilita a starilor (0 sau 1) pentru fiecare semnal in parte, functie de schema implementata la nivelul amplificatorului si de interfata utilizata pentru a citi aceste semnale de la amplificator.

Comenzile de la sistemul microprocesor catre amplificator vor fi, in cazul studiat:

- deblocarea franei electromagnetice;
- oprirea rapida a motorului;
- aprindere indicator OK;
- comanda contactor forta;
- aprindere bec eroare.

Aceste comenzi vor fi implementate ca biti ai unui port de iesiri numerice ale sistemului microprocesor, fiecare cu semnificatia indicata de numele semnalului, cu o conventie prestabilita a starilor (**0** sau **1**) pentru fiecare semnal in parte, functie de schema implementata la nivelul amplificatorului si de interfata utilizata pentru a genera aceste semnale de comanda.

- **Elementele de transmisie** vor fi utilizate in vederea adaptarii caracteristicilor motorului cu cele ale sarcinii (turatii, cupluri), si a tipului de miscare (rotativa, liniara). Parametrii necesari a fi considerati vor fi puterea maxima, turatia maxima, inertia si randamentul transmisiei. In principiu, acestia vor interveni in dimensionarea motorului prin modificarea parametrilor echivalenti (*moment de inertie*, *cuplu de sarcina* echivalent) totali ai sistemului, raportati la axul motorului.

6.1.9 Alegerea traductoarelor

Pentru masurarea diferitelor marimi din sistemul de actionare, vor fi utilizate traductoare si interfete specializate aferente, care vor permite achizitionarea acestor parametri de catre sistemul microprocesor. Pentru aplicatia propusa, vor fi necesare doua traductoare:

- **Traductorul de pozitie**, pentru a furniza masura pozitiei sistemului de actionare (montat pe motor sau/si pe sarcina). Uzual, o solutie ce ofera un raport avantajos pret/performanta este oferit de utilizarea traductoarelor de tip encoder incremental. Acesta este un traductor relativ, furnizand un numar specificat de impulsuri pentru o rotatie a axului sau. Acest numar de impulsuri pe rotatie va reprezenta rezolutia traductorului, care poate fi considerat ca un convertor analog/digital de pozitie. Astfel, pentru un numar de N impulsuri pe rotatie, parametrii echivalenti ai traductorului de pozitie vor fi:

- plaja de variatie a marimii de iesire: N [biti];

- rezolutia traductorului: $\lceil \log_2 N \rceil$ [biti];
- precizia traductorului: $2\pi/N$ [rad/bit].

Deoarece encoderul nu furnizeaza o informatie de pozitie absoluta, va fi utilizata o interfata specializata, in principiu un circuit de numarare, pentru a contoriza impulsurile furnizate de encoder. Totodata, deoarece utilizarea unui singur tren de impulsuri nu permite decelarea sensului de rotatie, encodelele furnizeaza 2 trenuri de impulsuri dreptunghiulare, decalate la 90° unul fata de celalalt. Un circuit detector de sens poate fi proiectat pentru a putea, pe baza sesizarii acestui defazaj (avans sau intarziere cu 90° a unui semnal fata de celalalt), sa se determine incrementarea sau decrementarea informatiei de pozitie. Deoarece acest traductor este un traductor relativ, care nu indica decat modificarile de pozitie fata de pozitia initiala a sistemului, encodelele furnizeaza inca un semnal numeric, asa-numitul semnal de zero al encoderului. Acest semnal devine activ (se emite un puls pe durata unui pas de encoder), doar o data pe o rotatie si poate fi folosit pentru initializarea schemei dupa punerea sub tensiune, pentru a se seta corespunzator o informatie de pozitie absoluta a sistemului. Figura 6.2 prezinta forma semnalelor furnizate de encoder (A si B, cele doua canale de masura; Z, canalul de pozitie initiala).

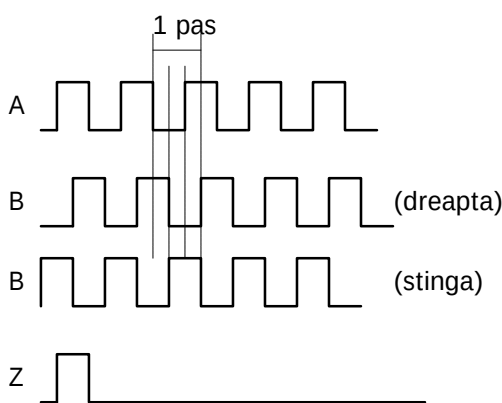


Figura 6.2 Semnalele generate la iesirea unui encoder de pozitie

De remarcat, de asemenea, ca este posibila utilizarea fiecaruia dintre fronturile crescatoare sau cazatoare ale semnalelor A si B ale encoderului, ceea ce permite marirea de 4 ori a preciziei encoderului, prin implementarea unor scheme de decodificare adecvate,.

Astfel, pentru exemplul abordat se considera ca se utilizeaza un encoder tipic cu un numar de 1024 de pulsuri pe rotatie; encoderul poate fi echivalat unui convertor A/D cu plaja de variatie de 1024 biti/rotatie, rezolutia de $\log_2 1024 = 10$ biti si precizia de $360^\circ/1024 \approx 0.3516^\circ/\text{bit}$. Utilizarea schemei de multiplicare cu 4 va creste rezolutia la 12 biti, iar precizia traductorului va ajunge la $\approx 0.088^\circ/\text{bit}$.

- **Traductorul de viteza** va furniza informatia de viteza a motorului; in cazul aplicatiei propuse, s-a presupus deja ca se utilizeaza un tahogenerator de c.c. pentru a furniza informatia de viteza necesara. Deoarece acest traductor furnizeaza o tensiune analogica la iesirea sa, proportionala cu viteza motorului, va fi necesara utilizarea unui convertor A/D pentru a obtine valoarea numerica a acestui semnal, utilizabila in sistemul microprocesor. Din cunoasterea valorii constantei tahogeneratorului, k_{TG} [V/rad/s], si a valorii maxime admise a vitezei motorului, $\Omega_{\max}$ [rad/s], se poate determina valoarea maxima a tensiunii de la iesirea tahogeneratorului pe care schema va trebui sa o masoare, $U_{TG\max} = k_{TG} * \Omega_{\max}$ [V]. Uzual, schema de conectare a iesirii tahogeneratorului la intrarea convertorului A/D va trebui adaptata (prin utilizarea unui simplu divizor rezistiv), pentru a adapta plaja de variatie a acestui semnal la plaja de valori acceptata de intrarea convertorului A/D (de

obicei, ± 10 sau ± 5 V). Rezolutia convertorului in aplicatiile de tipul celei studiate este de 10 sau 12 biti, uzual. Presupunem ca o rezolutie de 10 biti a fost decisa pentru masurarea vitezei sistemului, dedusa din valoarea vitezei minime pe care, prin tema de proiectare, utilizatorul o impune pentru sistem.

6.1.10 Alegerea tipului de schema de reglare

Pentru implementarea unui sistem de pozitionare, exista mai multe structuri posibile, diferite in functie de modul de realizare a controlului diferitelor variabile din sistem. Astfel, se pot in general distinge urmatoarele posibilitati:

- realizarea unui *sistem exclusiv discret de control*, pentru care ansamblul amplificator-motor-sarcina este condus in totalitate de sistemul numeric. Toate marimile de stare (*curent, viteza, pozitie*) sunt controlate de microprocesor. Aceasta solutie prezinta dezavantajul de a necesita componente discrete foarte rapide (in primul rand, microprocesorul), ceea ce conduce la sisteme foarte scumpe sau chiar, in unele cazuri, imposibil de realizat cu microprocesoare de tip general.
- realizarea unui *sistem partial discret de control*, pentru care sistemul numeric controleaza doar unele dintre variabilele de stare. Aceasta solutie ofera, pe de o parte, avantajul utilizarii unor bucle de control analogice pentru controlul unora dintre variabilele de stare, care au o variatie rapida, dar nu necesita implementarea unor bucle sofisticate de control — de exemplu, bucla de curent. Pe de alta parte, sistemul numeric va realiza reglarea numerica a celorlalte variabile de stare, ca viteza si pozitia, care nu au in mod uzual o dina-

mica atat de ridicata ca bucla de curent, dar necesita proceduri de reglare mai sofisticate.

Aceasta a doua solutie este preferabila pentru structura propusa in exemplul abordat. Figura 6.3 prezinta schema de principiu a sistemului in acest caz. Dupa cum se remarca, bucla de reglare a curentului este inchisa la nivelul amplificatorului, iar buclele de viteza si pozitie, la nivelul sistemu-

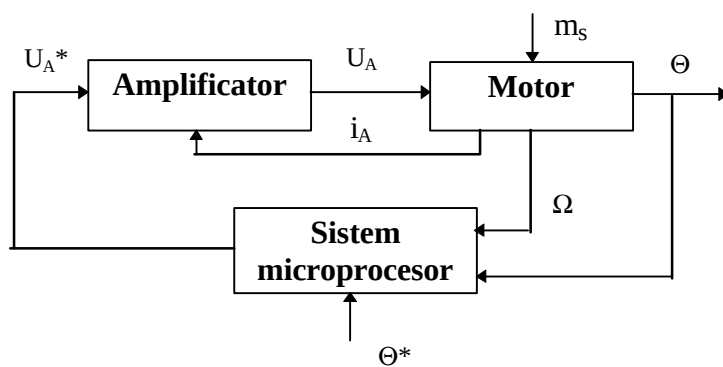


Figura 6.3 Schema sistemului de reglare a pozitiei cu bucla locala de curent si bucle digitale de viteza si pozitie

lui microprocesor.

6.1.11 Alegerea microprocesorului

Pe baza definirii caracteristicilor si a alegerii componentelor sistemului de pozitionare facute in paragrafele precedente, se poate trece la etapa definirii performantelor necesare pentru microprocesorul utilizat in sistemul de control. Pentru exemplul studiat, interfetele utilizate

pentru traductoarele de viteză și poziție, cât și interfața utilizată pentru generarea tensiunii de comandă a amplificatorului au o rezoluție de peste 8 biți. Totodată, din estimarea preciziei de calcul pentru implementarea procedurilor de control, se evidențiază necesitatea utilizării unei reprezentări matematice pe mai mult de 8 biți. Astfel rezultă în mod implicit că, pentru a se putea implementa în mod eficient sistemul de control propus, va fi nevoie să se utilizeze un microprocesor de 16 biți. Totodată, considerând că sistemul electromecanic motor-transmisie-sarcină conduce la obținerea unei constante de timp de ordinul 10 ms, conform teoremei esanționării a lui Shannon perioada minimă de esanționare va trebui să fie de 5 ms (frecvența de esanționare trebuie să fie de cel puțin două ori mai mare decât frecvența componentei de frecvență maximă a semnalului esanționat). Din considerente practice, însă, se preferă utilizarea unei perioade de esanționare de 5–10 ori mai mici — deci fie, în cazul nostru, **$h = 1$ ms**. Ținând cont că, pentru a asigura obținerea unor rezultate corecte prin schema de control discret, este necesar ca durata întregului proces de achiziție de date, calcul și comandă a amplificatorului să nu depășească un procent de maxim 10–30% din perioada de esanționare (altfel, trebuind să se considere în etapa de proiectare a schemei de control discret și timpul mort introdus de aceste întârzieri), deducem că sistemul microprocesor va trebui să realizeze aceste operații într-un interval de timp de aproximativ 100–300 μ s. Dacă se face un calcul estimativ, se poate deduce că, pentru un microprocesor de tip 8086 functionând la o frecvență de 8 MHz, durata medie a unei instrucțiuni este de 1 μ s, deci ar fi necesar ca rutina de control a sistemului să fie implementată prin executia a 100–300 de instrucțiuni. Aceste valori pot, în principiu, să permită implementarea unei bucle de control, fără a avea însă facilități deosebite de genul control adaptiv, optimizări etc. Dacă se doresc asemenea caracteristici pentru sistem, va fi necesar să se opteze pentru un sistem de performanțe superioare, soluția cea mai indicată fiind, la ora actuală, utilizarea unui microprocesor de prelucrare a semnalelor, de tip DSP.

Considerând că pentru aplicația studiată se va implementa o schemă de reglare cu parametri constanți, fără facilități de tipul celor sus-menționate, putem conchide, din analiza de mai sus, că soluția optimă performanță/preț pentru această aplicație va fi de a utiliza un microprocesor de tipul INTEL 8086.

6.2 PROIECTAREA SCHEMEI DE CONTROL. ASPECTE DE PRINCIPIU.

Paragrafele precedente au indicat modul de alegere a componentelor sistemului de poziționare și au indicat criteriile esențiale în ceea ce privește obținerea parametrilor reprezentativi, din punctul de vedere al interfatării acestora cu sistemul de control discret. Odată stabilită configurația sistemului și alese componentele sale, se poate trece la etapa obținerii **modelului sistemului** controlat (ansamblul *motor-amplificator-sarcină*). Pe baza modelului obținut și a performanțelor impuse sistemului controlat, se va putea alege structura și se vor putea calcula parametrii **schemei de control** discret utilizate.

Din analiza metodelor de control ce pot fi implementate pentru sistemul de poziționare, o soluție ce oferă anumite avantaje în ceea ce privește performanțele obținute și robustețea schemei de control este cea a implementării unei scheme de reglare cu reacție după stare, calculul coeficienților schemei de reglare fiind efectuat printr-o procedură de alocare a polilor.

6.2.1 Obținerea modelului sistemului

În vederea implementării, după cum s-a menționat, a unei scheme de reglare cu reacție după stare, pentru controlul poziției unui servomotor de c.c. este necesară obținerea modelului de stare al ansamblului amplificator-motor-sarcină.

Astfel, într-o reprezentare având ca variabile de stare curentul, viteza și poziția motorului, deci pentru care vectorul de stare este dat de relația

$$x = \begin{Bmatrix} i_A \\ \Omega \\ \Theta \end{Bmatrix}, \quad (6.3)$$

setul de ecuații de stare al motorului și al sarcinii raportate la axul acestuia va fi:

$$\begin{bmatrix} \dot{i}_A \\ \dot{\Omega} \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} 1/T_A & K_E \Phi_E & 0 \\ K_E \Phi_E / J & -1/T & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} i_A \\ \Omega \\ \Theta \end{bmatrix} + \begin{bmatrix} 1/L_A \\ 0 \\ 0 \end{bmatrix} u_A + \begin{bmatrix} 0 \\ 1/J \\ 0 \end{bmatrix} m_s \quad (6.4)$$

unde J = momentul total de inerție raportat la axul motorului, T = constanta electromecanică totală echivalentă a sistemului.

Pentru ansamblul *amplificator-motor-sarcină* însă, bucla de curent fiind încorporată în structura analogică de reglare a amplificatorului, considerând că mărimea de comandă a acestuia este **curentul impus** (deci, considerând că se utilizează amplificatorul ca sursă de curent a motorului), schema echivalentă a sistemului controlat devine cea din figura 6.4.

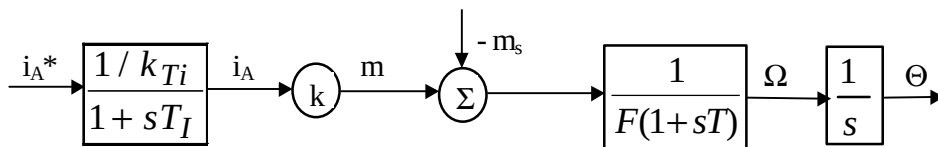


Figura 6.4 Schema ansamblului amplificator de curent-motor-sarcină

Pentru această schemă, considerând neglijabilă valoarea constantei de timp a amplificatorului T_I , (ipoteză valabilă pentru cazurile practice), modelul de stare va fi dat de ecuațiile:

$$\begin{bmatrix} \dot{\Omega}_T \\ \dot{\Theta} \end{bmatrix} = \begin{bmatrix} -1/T & 0 \\ 1/k_{T\Omega} & 0 \end{bmatrix} \begin{bmatrix} \Omega_T \\ \Theta \end{bmatrix} + \begin{bmatrix} \frac{k \cdot k_{T\Omega}}{J \cdot k_{TI}} \\ 0 \end{bmatrix} i_A^* + \begin{bmatrix} \frac{-k_{T\Omega}}{J} \\ 0 \end{bmatrix} m_s \quad (6.5)$$

unde s-au considerat parametrii echivalenți ai schemei, incluzând și constantele traductoarelor pentru traductorul de viteză $k_{T\Omega} = \Omega_T / \Omega_{\max}$ (Ω_T = ieșirea tahogeneratorului), iar pentru bucla de amplificare, $k_{TI} = I_{A\max} / I_{A\max}^*$.

Asadar, setul de ecuatii de stare ale ansamblului *amplificator-motor-sarcina* poate fi reprezentat sub forma clasica a ecuatiilor de stare, de forma:

$$\begin{aligned} \dot{x} &= A \cdot x + b \cdot u + e \cdot v \\ y &= c_T \cdot x \end{aligned} \quad (6.6)$$

unde $u \equiv i_A^*$, $v \equiv m_s$, $c_T = [0 \ 1]$.

Modelul astfel obtinut va fi utilizat in continuare pentru obtinerea modelului discret al sistemului si proiectarea controlerului cu reactie dupa stare.

6.2.2 Discretizarea modelului sistemului

In vederea discretizarii modelului continuu obtinut in paragraful anterior, este necesara cunoasterea pasului de esantionare cu care va lucra sistemul de control digital. Valoarea acestui parametru se obtine din performantele de regim dinamic impuse sistemului de pozitionare prin tema de proiectare, urmandu-se in esenta o metodologie axata pe urmatoarele considerente:

- se estimeaza timpul de crestere (timpul necesar pentru atingerea valorii 0.9 din valoarea impusa pentru referinta), $t_c \approx T$, constanta de timp principala a procesului controlat.
- se estimeaza timpul tranzitoriu (pentru atingerea benzii de +/-2% din valoarea de referinta), cu relatia $t_t \approx 4 t_c$.
- utilizand factorul de amortizare ξ (ales, uzual, de o valoare apropiata de 0.7), se determina pulsatia naturala a sistemului reglat, cu relatia $\Omega_n \approx 4/\xi t_t$.
- latimea de banda a sistemului se obtine din relatia $\Omega_b = 4\pi/t_t = \pi/T$.
- pulsatia de esantionare se alege cu relatia $\Omega_s \approx (5 \div 10)\Omega_b$.
- pasul de esantionare rezulta: $h = 2\pi/\Omega_s$ [s].

Astfel, se poate determina modelul discretizat al procesului, cu relatiile:

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{A}_d \mathbf{x}_i + \mathbf{b}_d u_i + \mathbf{e}_d v_i \\ y_i &= \mathbf{c}_d^T \mathbf{x}_i \end{aligned} \quad (6.7)$$

unde $\mathbf{x}_i$ = vectorul de stare la momentul i de esantionare, u_i = comanda la momentul i de esantionare, iar matricele $\mathbf{A}_d$, $\mathbf{b}_d$, $\mathbf{e}_d$ si $\mathbf{c}_d^T$ sunt date de relatiile:

$$\mathbf{A}_d = e^{A h}, \quad \mathbf{b}_d = \int_0^h e^{A t} \cdot b \cdot dt, \quad \mathbf{e}_d = \int_0^h e^{A t} \cdot e \cdot dt, \quad \mathbf{c}_d^T = c^T \quad (6.8)$$

De remarcat totodata ca, pe baza performantelor de regim dinamic impuse sistemului si pe baza pasului de discretizare, se vor putea determina si polii sistemului, atat in planul s cat si in planul z . Astfel:

- se determina polii principali ai sistemului reglat in planul s:

$$s_{1,2} = -x \cdot w_n \pm j \cdot w_n \cdot \sqrt{1-x^2}$$

- restul polilor se alocă uzual pe axa reala negativa, la distante de (5÷10) ori partea reala a polilor $s_{1,2}$.
- se calculeaza polii in planul z, cu relatia $z_k = e^{s_k \cdot h}$.

6.2.3 Proiectarea controlerului discret

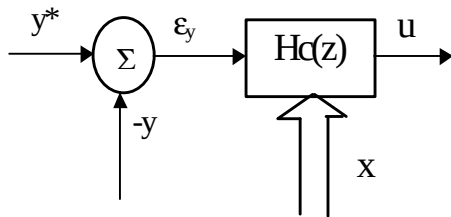


Figura 6.5 Schema de principiu a unui regulator dupa stare

Pe baza cunoasterii modelului sistemului, se poate trece la etapa proiectarii controlerului discret cu reactie dupa stare. Schema de principiu a unui asemenea controler este prezentata in figura 6.6. Marimea de comanda a sistemului, u (iesirea controlerului) va fi calculata pe baza unei relatii de forma:

$$u = \mathbf{f}^T \mathbf{x} \quad (6.9)$$

deci reprezinta o combinatie liniara a componentelor vectorului de stare al sistemului, coeficientii $\mathbf{f}^T$ reprezentand ponderile diverselor componente ale vectorului de stare $\mathbf{x}$ la calculul marimii u. La vectorul de stare al sistemului, mai trebuie adaugat si efectul tipului de variatie in timp al semnalului de referinta, conform principiului “*modelului intern*” al compensatorului. Astfel, pentru o referinta de tip treapta, schema de principiu a sistemului de reglare va deveni cea din figura 6.6. Dupa cum se poate remarca din figura, la vectorul de stare $\mathbf{x}$ se va adauga o variabila de stare suplimentara, ζ , care nu reprezinta altceva decat integrala marimii de eroare, ε . Astfel, la ecuatiile de stare ale sistemului se va adauga si ecuatia suplimentara:

$$\zeta_{i+1} = \zeta_i + \varepsilon_i \quad (6.10)$$

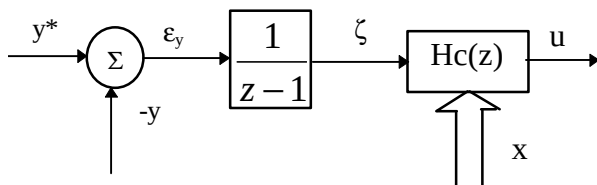


Figura 6.6 Structura de principiu a regulatorului dupa stare in cazul referintei treapta

deci ordinul n al sistemului de ecuatii de stare va creste cu o unitate, capatand valoarea $n_e = n + 1$. In consecinta, sistemul (6.7) de ecuatii de stare capata forma extinsa (6.11):

$$\begin{aligned} \mathbf{x}_{i+1}^e &= \mathbf{A}_{de} \mathbf{x}_{ie} + \mathbf{b}_{de} u_i + \mathbf{e}_{de} V_i \\ y_{ie} &= \mathbf{c}_{de}^T \mathbf{x}_i^e \\ u_i &= \mathbf{f}_{de}^T \mathbf{x}_i^e \end{aligned} \quad (6.11)$$

unde

$$\mathbf{x}_i^e = \begin{bmatrix} x_i \\ z_i \end{bmatrix}, \quad \mathbf{A}_{de} = \begin{bmatrix} \mathbf{A}_d & 0 \\ \mathbf{c}_d^T & 1 \end{bmatrix}, \quad \mathbf{b}_{de} = \begin{bmatrix} \mathbf{b}_d \\ 0 \end{bmatrix}, \quad \mathbf{e}_{de} = \begin{bmatrix} \mathbf{e}_d \\ 0 \end{bmatrix}, \quad \mathbf{c}_{de}^T = \begin{bmatrix} \mathbf{c}_d^T & 0 \end{bmatrix} \quad (6.12)$$

Coeficientii de comanda $\mathbf{f}_{de}^T$ vor fi calculati prin rescrierea sistemului (6.11)

$$\begin{aligned} \mathbf{x}_{i+1}^e &= (\mathbf{A}_{de} + \mathbf{b}_{de} \mathbf{f}_{de}^T) \mathbf{x}_i^e + \mathbf{e}_{de} v_i \\ y_{ie} &= \mathbf{c}_{de}^T \mathbf{x}_i^e \end{aligned} \quad (6.13)$$

Daca $\Lambda_e = \{ z_1, z_2, \dots, z_{ne} \}$ reprezinta multimea polilor complecsi obtinuti prin impunerea performantelor dinamice, prin efectuarea unei proceduri de alocare a polilor se asigura satisfacerea relatiei

$$\sigma(\mathbf{A}_{de} + \mathbf{b}_{de} \mathbf{f}_{de}^T) = \Lambda_e \quad (6.14)$$

de unde rezulta coeficientii $\mathbf{f}_{de}^T$. Astfel, se vor obtine termenii vectorului $\mathbf{f}_{de}^T = [f_1, f_2, f_3]$, iar comanda u_i se va obtine din relatia

$$u_i = f_1 \Omega_i + f_2 \Theta_i + f_3 \zeta_i \quad (6.15)$$

Figura 6.7 prezinta schema sistemului de reglare in acest caz, indicand atat structura sistemului reglat, cat si structura regulatorului dupa stare implementat in sistemul microprocesor.

Dupa cum se remarca din relatiile prezentate si din schema din figura 6.7, calculul marimii de comanda a amplificatorului se bazeaza pe o combinatie liniara a marimilor de stare (*viteza, pozitia si integrala erorii de pozitie*).

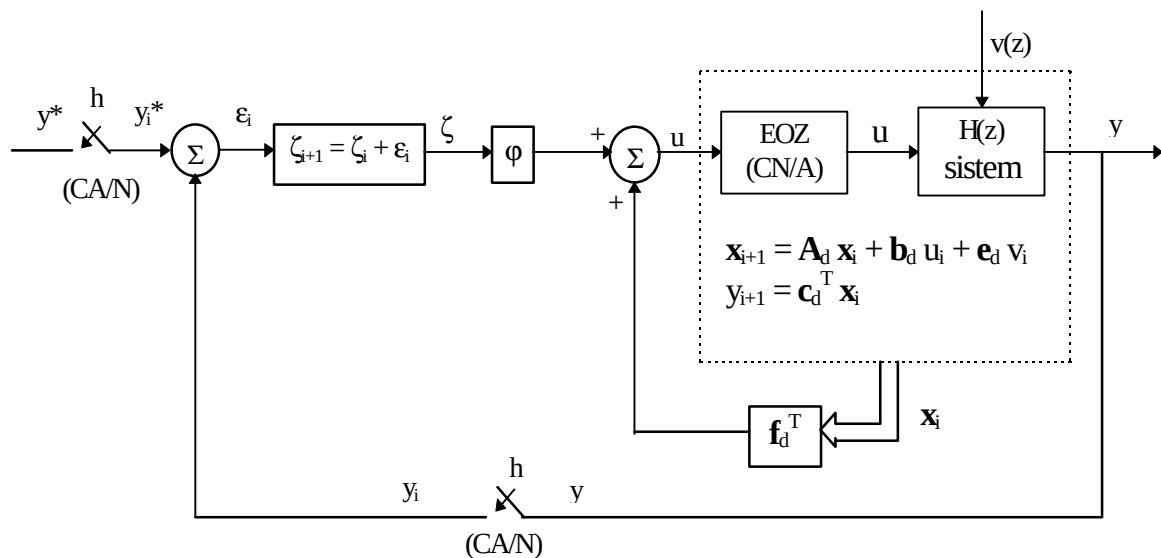


Figura 6.7 Schema de implementare a sistemului de reglare discreta dupa stare,

pentru semnal de referinta treapta

Implementarea practica a algoritmului de control discret se va prefera insa sub urmatoarea forma, care asigura o mai mare protectie impotriva eventualelor depasiri de calcul:

- se masoara variabilele de stare din sistem la momentul i : viteza Ω_i si pozitia, Θ_i .
- se calculeaza eroarea de pozitie la momentul i : $\varepsilon_i = \Theta_i^* - \Theta_i$
- se calculeaza variatia pozitiei fata de momentul $i-1$: $\delta\Theta_i = \Theta_i - \Theta_{i-1}$
- se calculeaza variatia vitezei fata de momentul $i-1$: $\delta\Omega_i = \Omega_i - \Omega_{i-1}$
- se calculeaza comanda la momentul i : $u_i = u_{i-1} + f_1 \delta\Omega_i + f_2 \delta\Theta_i + f_3 \varepsilon_i$

6.3 PROIECTAREA STRUCTURII HARDWARE A SISTEMULUI

Paragrafele precedente au permis definirea tuturor cerintelor aplicatiei si definirea componentelor utilizate pentru a se configura intregul sistem de pozitionare. Pe baza cunoasterii performantelor cerute sistemului de control numeric, a evaluarii capacitatii de calcul si a vitezei acestuia, s-au putut indica tipul si caracteristicile definitorii ale microprocesorului ce trebuie folosit si se poate estima necesarul de memorie program si de date pentru implementarea aplicatiei propuse. De asemenea, pe baza cunoasterii structurii partii de forta, a interfetelor ce trebuiesc realizate pentru achizitia de date din sistem, comanda elementului de executie, si elementele de diagnoza, si estimand elementele de interfatare cu utilizatorul sistemului de comanda, se vor putea estima si principalele cerinte in ceea ce priveste necesarul si tipul interfetelor de intrare si de iesire ale sistemului.

Odata specificat in intregime necesarul de resurse ale sistemului microprocesor, se va putea incepe proiectarea propriu-zisa a structurii hardware a acestuia.

Vom trece la evaluarea acestui necesar de resurse, in ceea ce priveste structura aleasa pentru sistemul microprocesor, memoria PROM si RAM, cat si interfetele de intrare si de iesire.

Vom considera ca se utilizeaza un microprocesor de tip **INTEL 8086**, functionand la o frecventa de 24 Mhz, configurat in modul minim (deci intr-o structura uniprocessor), asadar generand direct semnalele de control, fara a necesita utilizarea unui circuit controler de magistrala, de tip **INTEL 8288**. Pentru furnizarea semnalului de ceas **CLK** si a semnalelor **READY** si **RESET**, se utilizeaza circuitul specializat **INTEL 8284** (v.cap.4). Conexiunile microprocesorului la circuitul de ceas (**8284**), circuitele latch pentru memorarea adreselor (**8282**) si circuitele tampon pentru magistrala de date (**8286**) sunt prezentate in schema de conexiuni din figura 6.8. Bitii magistralelor de adrese (A19–A0) si ai celei de date (D15÷D0), cat si semnalele de control generate de microprocesor, vor fi utilizate pentru conectarea cu toate resursele sistemului (memorii si porturi).

In ceea ce priveste necesarul de memorie PROM, in care se va stoca programul ce controleaza sistemul, evaluarea acestuia se va face pe baza experientei prealabile a

proiectantului, combinata cu eventuala evaluare bazata pe scrierea si executia pe un sistem de dezvoltare a principalelor module ale programului. Totodata, nu trebuie uitat ca, data fiind insasi "filosofia" proiectarii acestor sisteme, vor trebui estimate si lasate rezerve de memorie pentru eventuale dezvoltari ulterioare ale sistemului, adaugari de noi functii etc. Dupa cum s-a mentionat in cateva randuri deja, aceste modificari vor implica, in cele mai multe cazuri, modificarea sau adaugarea de noi parti de program.

Bazat pe aceste elemente, pentru aplicatia propusa si in ideea lasarii unei rezerve pentru extinderea programului la functii suplimentare care nu vor fi incluse in versiunea ce se va prezenta aici, se poate estima ca o cantitate de 16 ko de memorie **PROM** va fi suficienta. Consideram ca se utilizeaza circuite de tipul **2716**, cu o capacitate de 2k x 8 pe cip. Asadar, necesarul de memorie PROM va fi asigurat cu un numar de 8 cipuri.

Din considerente similare, necesarul de memorie **RAM** este estimat la 4 ko. Consideram ca se utilizeaza circuite de tip **2114**, cu o capacitate de 1k x 4 pe cip, deci vor fi necesare 8 cipuri de acest tip.

- In ceea ce priveste **porturile** sistemului, vor fi necesare:

– **porturi de intrare:**

- 1 port: interfata traductorului de pozitie - encoder / circuite de numarare (10 biti);
- 1 port: interfata traductorului de viteza - tahogenerator / convertor A/D (10 biti);
- 1 port: semnalizari generale (16 biti);
- 1 port: tastatura locala - linii tastatura (16 biti);

– **porturi de iesire:**

- 1 port : comanda amplificatorului de curent - convertor D/A (10 biti);
- 1 port : tastatura locala - selectii coloane taste (16 biti);
- 1 port : comenzi digitale generale (16 biti);
- 2 porturi : afisaj local (4 cifre a 8 segmente fiecare) (2x16 biti).

– **porturi speciale:**

- circuit de ceas programabil **INTEL 8253**;
- controler de intreruperi **INTEL 8259A**.

Presupunem urmatoarea **localizare a resurselor** sistemului:

- memoria PROM, in ultimii **16 ko** de memorie ai sistemului: zona de adrese (**0FC000h ÷ 0FFFFFFh**);
- memoria RAM, in primii **4 ko** de memorie ai sistemului: zona de adrese (**0h ÷ 0FFFh**);
- porturile I/O: incepand de la adresa **300h** (compatibile cu adresele rezervate pentru interfete utilizator in calculatoarele PC).

In continuare, se vor prezenta considerentele de proiectare, tabelele de decodificare si schemele de implementare pentru toate resursele sistemului microprocesor, atat memorie cat si porturi.

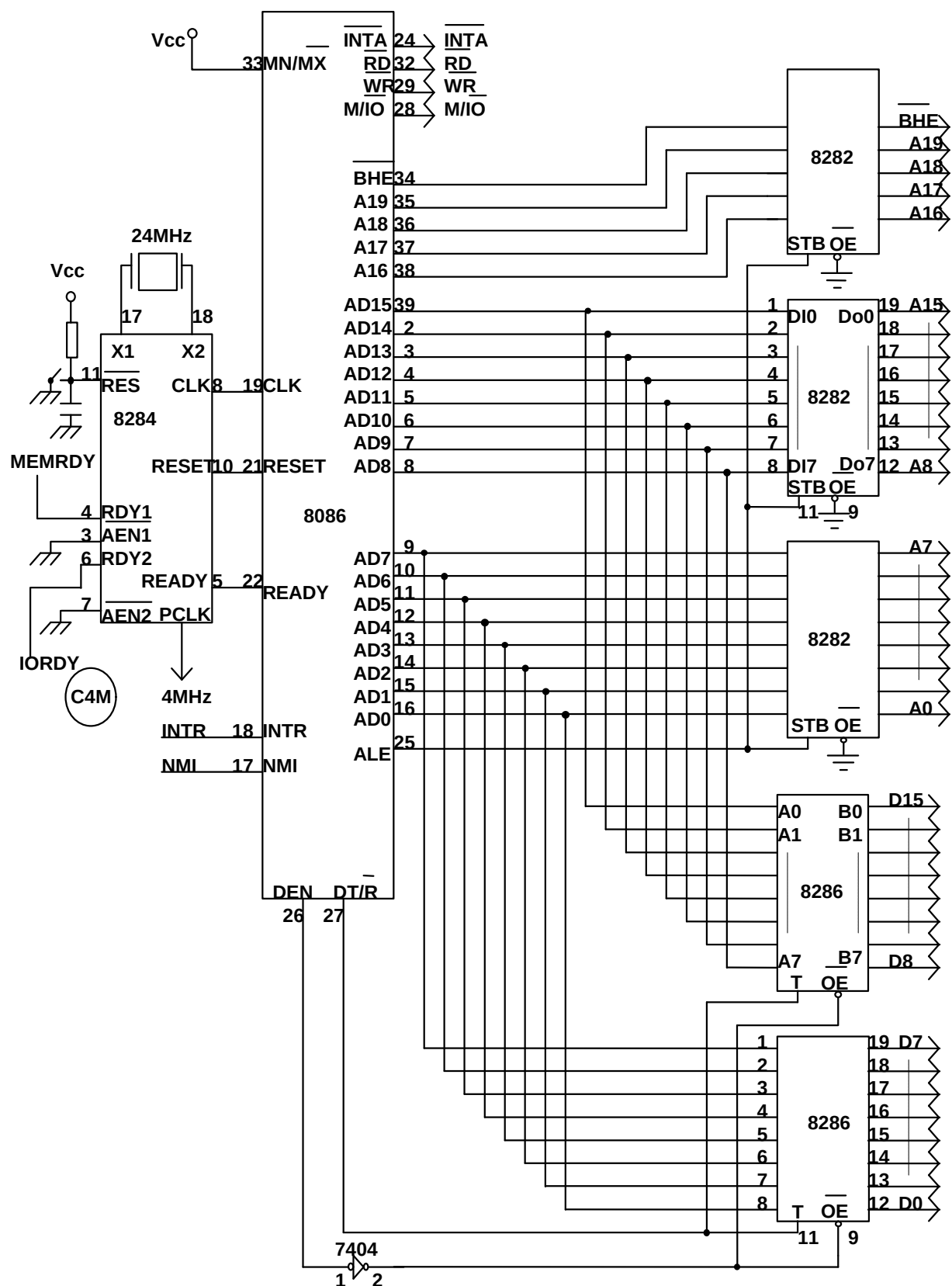


Figura 6.8 Schema de conexiuni a microprocesorului 8086

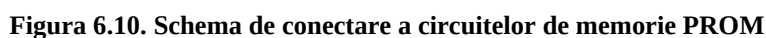
6.3.1 Memoria PROM

Dupa cum s-a precizat deja, se va implementa un total de 16 ko memorie PROM, utilizand circuite de tip 2716, cu o capacitate de $2k \times 8$ pe cip. Adresa de inceput a zonei de memorie PROM este 0FC000h. Zona de memorie ocupata va fi, pana la sfarsitul memoriei microprocesorului, 0FFFFFFh. Cipurile vor fi conectate astfel: 4 in blocul de memorie de la adrese pare si 4 in blocul de memorie de la adrese impare. Avand o capacitate de 2 ko, cipurile au un numar de 11 biti de adrese ($2k = 2048 = 2^{11}$), deci bitii de adrese A1–A11 vor fi conectati direct pe cipuri. Bitul A0 si semnalul $\overline{BHE}$ vor fi folosite pentru selectia cipurilor de la adrese pare, respectiv impare. Bitii de adrese superioare, A12–A19, vor fi utilizati pentru a obtine, prin decodificare, semnalele de selectie de cip ale circuitelor ($\overline{CS}$). Tabelul de decodificare 6.1 prezinta modul de obtinere a semnalelor de selectie de cip pentru toate memoriile PROM ale schemei. Se remarca utilizarea, pentru generarea acestor selectii, si a semnalelor de control $\overline{M/\overline{IO}}$ si $\overline{RD}$ (validand selectia doar pentru operatii de citire din memorie). Circuitele PROM sunt denotate P0 la P7, indicandu-se si zona de adrese ocupata de catre fiecare dintre ele.

Tabelul 6.1. Alocarea memoriei pentru selectia circuitelor PROM

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	...	...	A 2	A 1	A 0	$\overline{BHE}$	$\overline{RD}$	$\overline{M/\overline{IO}}$	Port selectat P0-P7	Adrese
1	1	1	1	1	1	1	1	x	...	...	x	x	0	x	0	1	$\overline{P6}$	FF000÷FFFFE (pare)
1	1	1	1	1	1	1	1	x	...	...	x	x	0	x	0	1	$\overline{P7}$	FF001÷FFFFF (impare)
1	1	1	1	1	1	1	0	x	...	...	x	x			0	1	$\overline{P4\&P5}$	FE000÷FEFFF
1	1	1	1	1	1	0	1	x	...	...	x	x			0	1	$\overline{P2\&P3}$	FD000÷FDFFF
1	1	1	1	1	1	0	0	x	...	...	x	x			0	1	$\overline{P0\&P1}$	FC000÷FDFFF

Figura 6.9 prezinta modul detaliat de sinteza a semnalelor de selectie de cip a acestor circuite. Se remarca utilizarea unui circuit NAND cu 8 intrari pentru a genera semnalul de selectie E_3 pentru decodicatorul 8205, bazat pe combinatia bitilor de adrese superioare A19÷A14 si a semnalului $\overline{M/\overline{IO}}$. Aceste semnale vor valida operatiile de citire din pagina de memorie ocupata de cipurile de memorie PROM, pentru care bitii de adrese vor avea valorile: A19 = 1, A18 = 1, A17 = 1, A16 = 1, A15 = 1, A14 = 1, iar semnalul $\overline{M/\overline{IO}} = 1$ (conform tabelului 6.1, pentru toate cipurile de memorie PROM, acestea sunt valorile semnalelor respective ce valideaza selectia unuia dintre PROM-uri). Bitii de adresa A13 si A12 sunt apoi utilizati la nivelul decodicatorului, pentru a genera 4 semnale de selectie, corespunzatoare uneia dintre cele 4 pagini de cate 2 ko (1 k word) fiecare, in care se poate sub-imparti memoria PROM a sistemului. In fine, utilizand semnalele A0 si $\overline{BHE}$, se pot genera efectiv semnalele de selectie de cip, pentru fiecare dintre cele 8 cipuri de memorie PROM, $\overline{CSP0}$ la $\overline{CSP7}$. A0 valideaza generarea selectiilor de cip pentru cipurile de la adresele pare, iar $\overline{BHE}$ pentru cipurile de la adresele impare (v. cap.4, pentru detalii relativ la mecanismul adresarii memoriei in sistemele cu microprocesor 8086).



Tabelul 6.2. Alocarea memoriei pentru selectia circuitelor RAM

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	...	A 2	A 1	A 0	BHE	M/IO	SEL	Adresse
0	0	0	0	0	0	0	0	0	x	...	x	x	0	x	1	CSR0	0 - 007FF
													x	0		CSR1	
0	0	0	0	0	0	0	0	1	x	...	x	x	0	x	1	CSR2	00800 - 00FFF
													x	0		CSR3	

6-22

si a semnalului $\overline{M/\overline{IO}}$. Aceste semnale vor valida operatiile de citire din pagina de memorie ocupata de cipurile de memorie RAM pentru care bitii de adrese vor avea valorile: $A_{19} = 0$, $A_{18} = 0$, $A_{17} = 0$, $A_{16} = 0$, $A_{15} = 0$, $A_{14} = 0$, $A_{13} = 0$, $A_{12} = 0$, iar semnalul $\overline{M/\overline{IO}} = 1$ (conform tabelului 6.2, pentru toate cipurile de memorie RAM, acestea sunt valorile semnalelor respective ce valideaza selectia unei perechi de 2 dintre RAM-uri). Bitul de adresa A_{11} este apoi utilizat pentru a genera 2 semnale de selectie, corespunzatoare uneia din cele 2 pagini de cate 2 ko (1 k word) fiecare, in care se poate subimparti memoria RAM a sistemului. In fine, utilizand semnalele A_0 si $\overline{BHE}$, se pot genera efectiv semnalele de selectie de cip pentru fiecare din cele 8 cipuri de memorie RAM, $\overline{CSR0}$ la $\overline{CSR3}$. A_0 valideaza generarea selectiilor de cip pentru cipurile de la adresele pare, iar $\overline{BHE}$ pentru cipurile de la adresele impare (v. cap. 4, pentru detalii relativ la mecanismul adresarii memoriei in sistemele cu microprocesor 8086).

De remarcat, din figura 6.11, utilizarea semnalului de selectie a paginii de memorie RAM si pentru a genera semnalul $\overline{RAMRDY}$, semnal care este fi utilizat la generarea semnalului $\overline{READY}$ al microprocesorului. Acest semnal este utilizat pentru a genera $\overline{READY}$ ori de cate ori va avea loc o operatie de selectie a memoriei RAM a sistemului, deoarece timpul de acces la aceste circuite este suficient de redus (300 ns) pentru a nu necesita introducerea unor perioade de asteptare in ciclul de acces la memoria respectiva.

Semnalul $\overline{RAMRDY}$ este fi combinat cu semnalul $\overline{PROMRDY}$, dupa cum se remarca din figura, pentru a genera semnalul $\overline{MEMRDY}$, indicand operatie valida cu memoria sistemului (PROM sau RAM), in cazul in care s-a adresat o celula de memorie din zona de PROM sau din cea de RAM a sistemului microprocesor. Acest semnal va fi utilizat la nivelul circuitului 8284 pe pinul $\overline{RDY1}$, pentru a genera semnalul $\overline{READY}$ catre procesor.

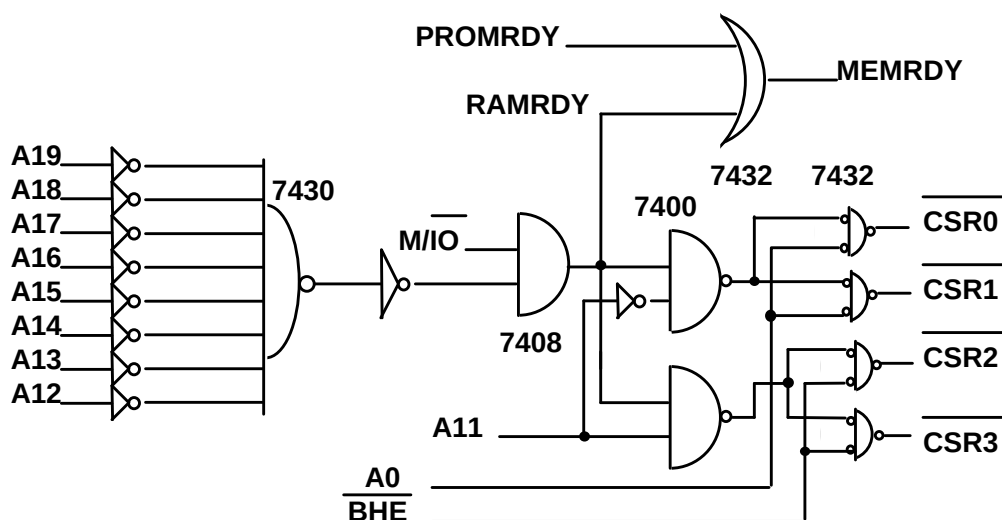


Figura 6.11. Schema de decodificare pentru memoriile RAM

In fine, figura 6.12 prezinta schemele de conexiuni ale cipurilor de memorie RAM ale sistemului. Se remarca utilizarea bitilor de adrese $A_{10} \div A_1$ direct pe fiecare cip. Semnalul

$\overline{WR}$ valideaza operatiile de scriere in memorie, prin conectare la pinul de control de activare a scrierii, $\overline{WE}$. Iesirile de date ale memoriilor sunt conectate pentru cipurile de la adresele pare ($\overline{CSR0}$ si $\overline{CSR2}$) la bitii inferiori ai magistralei de date (D7 la D0), iar pentru cele de la adresele impare ($\overline{CSR1}$ si $\overline{CSR3}$), la bitii superiori ai magistralei de date (D15 la D8). Se remarca utilizarea a cate 2 cipuri selectate simultan, pentru a putea genera cuvinte de date de cate 8 biti, datorita faptului ca cipurile de memorie RAM au cuvinte de date de cate 4 biti fiecare.

6.3.3 Porturile de intrare-iesire

Din analiza necesarului de porturi de intrare-iesire prezentata la inceputul paragrafului 6.3, se poate alcatui tabela de selectie 6.3. Aceasta permite generarea a 8 grupe de selectie de cate 8 porturi fiecare, asigurand astfel o structura unitara pentru grupele de porturi existente in sistem. Desi nu toate porturile astfel configurate sunt efectiv utilizate in implementarea prezentata, se asigura omogenitatea schemei de decodificare, ceea ce reduce in mod simtitor gradul de complexitate a schemei respective. Se vor considera doar porturi care se gasesc la adrese pare, astfel se pot adresa atat porturi pe un octet, cat si porturi pe un cuvant de 16 biti, utilizand aceleasi semnale de selectie. Bitii de adresa A1 si A2 vor putea fi folositi in interiorul fiecareia din paginile de port astfel generate, pentru eventuale decodificari ulterioare. Considerand adresa de baza a zonei de porturi 300h, selectiile paginilor de port (semnalele $\overline{P0} \div \overline{P7}$) vor fi generate la aceleasi valori ale bitilor superiori de adrese ($A15=0$, $A14=0$, $A13=0$, $A12=0$, $A11=0$, $A10=0$, $A9=1$, $A8=1$, $A7=0$, $A6=0$) si la operatii de selectie a porturilor, $M/\overline{IO}=0$. Bitii de adrese A5, A4 si A3 vor genera cele 8 combinatii corespunzatoare celor 8 pagini de port configurate.

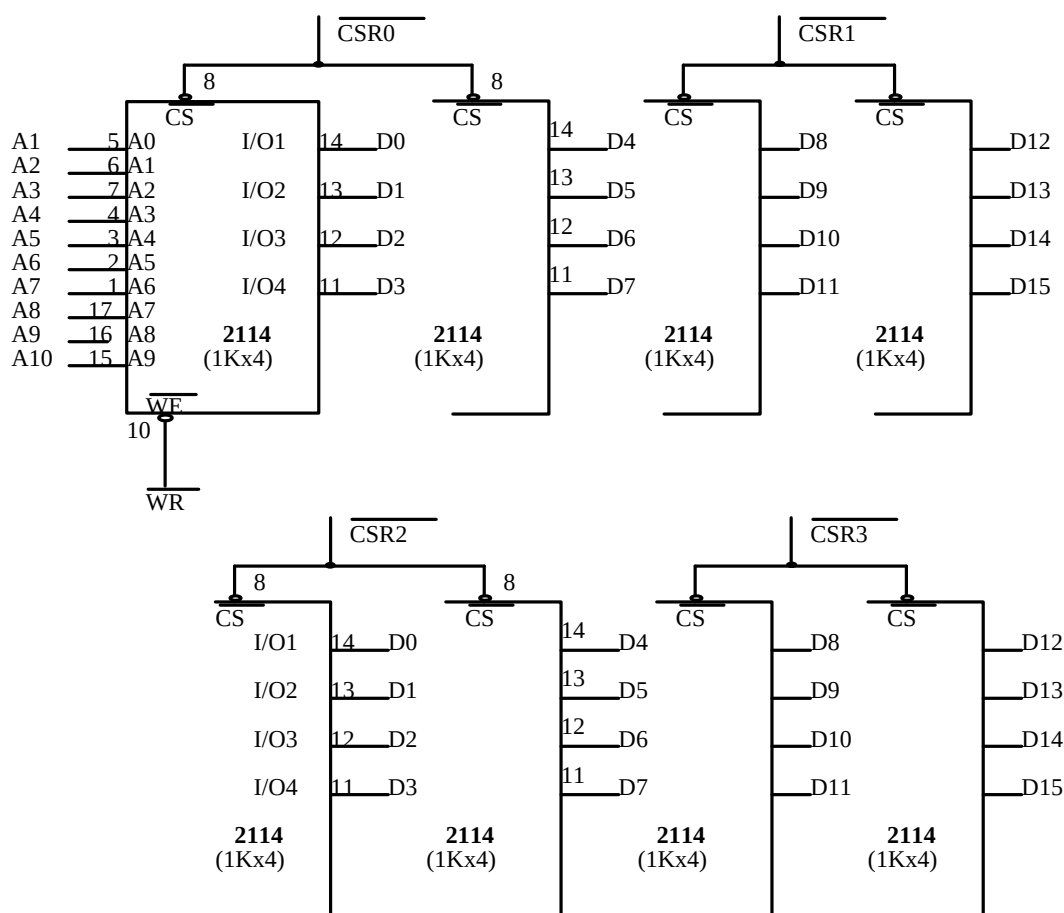


Figura 6.12. Schema de conectare a circuitelor de memorie RAM

Tabelul 6.3 prezinta si zonele de adrese ocupate de fiecare dintre cele 8 pagini de port.

Tabelul 6.3. Alocarea adreselor pentru selectia porturilor sistemului

A 15	A 14	A 13	A 12	A 11	A 10	A 9	A 8	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0	M/I/O	SE L	Adrese
0	0	0	0	0	0	1	1	0	0	0	0	0	x	x	x	0	$\overline{P0}$	300 - 307
										0	0	1					$\overline{P1}$	308 - 30F
										0	1	0					$\overline{P2}$	310 - 317
										0	1	1					$\overline{P3}$	318 - 31F
										1	0	0					$\overline{P4}$	320 - 327
										1	0	1					$\overline{P5}$	328 - 32F
										1	1	0					$\overline{P6}$	330 - 337
										1	1	1					$\overline{P7}$	338 - 33F

Figura 6.13 prezinta schema de implementare a adresarii globale a porturilor si de generare a semnalelor $\overline{P0}$ ÷ $\overline{P7}$ in conformitate cu continutul tabelii 6.3 si a considerentelor prezentate

mai sus. Se remarca utilizarea tuturor semnalelor de selectie ale circuitului decodificator 8205, pentru a valida diversele combinatii de semnale necesare in acest scop. Totodata, similar situatiei de la adresarea memoriilor, acelasi semnal care delimiteaza operatiile cu porturi din pagina globala de $8 \times 8 = 64$ porturi configurata in sistem se utilizeaza si pentru generarea unui semnal IORDY, indicand microprocesorului efectuarea unei operatii valide cu unul dintre porturile definite ale sistemului. Acest semnal va fi utilizat la nivelul circuitului 8284 pe pinul RDY2, pentru a genera semnalul READY catre procesor.

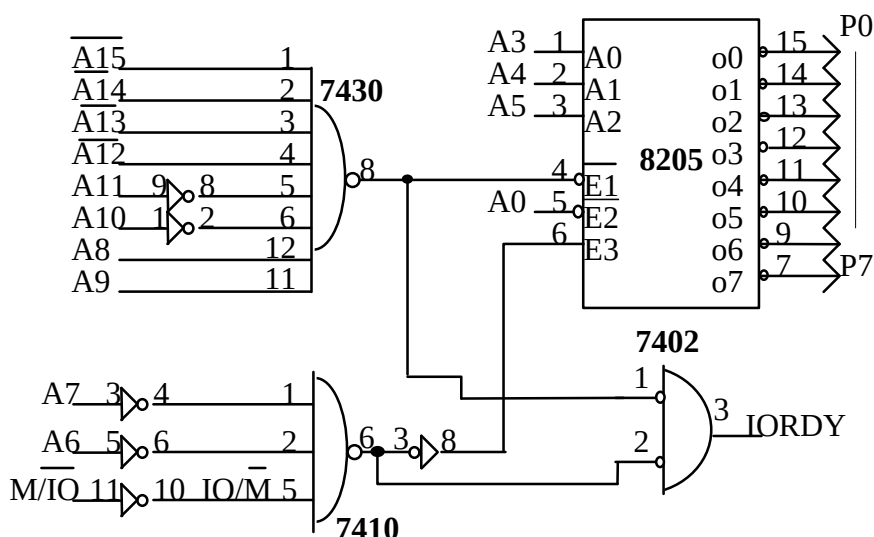


Figura 6.13. Schema de decodificare pentru porturile de intrare-iesire

In paragrafele urmatoare se vor utiliza in mod diferentiat aceste semnale globale de selectie, pentru a implementa in mod specific fiecare dintre porturile sistemului.

6.3.3.1 Interfata traductorului de pozitie

Traductorul de pozitie, encoder incremental cu 1024 de pulsuri pe tura, va trebui conectat la microprocesor prin intermediul unei interfete dedicate, proiectata pentru a furniza, pe baza pulsurilor de la encoder, informatia de pozitie a sistemului. Figura 6.14 prezinta o posibila structura pentru a implementa aceasta interfata. Semnalele A si B furnizate de catre encoder (v. par. 6.1.9 pentru detalii privind functionarea encoderului), sunt aplicate ca semnale de intrare intr-un circuit care detecteaza sensul de miscare si apoi determina contorizarea impulsurilor primite, pentru a stoca pozitia absoluta a masinii, pe o rotatie a acesteia. Semnalul Z de la encoder este utilizat pentru a initializa cu 0 continutul numaratorului ce contine informatia de pozitie. Acest numarator are 10 biti (corespunzator numarului de 1024 de pulsuri generate de catre encoder pentru o rotatie completa) si este reversibil. In functie de sensul de deplasare (dreapta/stanga), continutul sau se va incrementa, respectiv decrementa. Avand doar 10 biti, informatia de pozitie este absolut cunoscuta doar pentru o tura a axului encoderului. Va fi necesar ca pentru deplasari mai mari de o rotatie sa se contorizeze numarul de ture parcurse prin program, deci la nivel software. Din acest moment, in functie de tipul variabilei utilizate in program pentru a memora numarul de ture efectuate, plaja de valori

absolute a pozitiei poate fi oricat de mare (de exemplu, pentru o variabila de tip intreg reprezentata pe 16 biti, poate fi memorata o plaja de pozitii cuprinsa intre -32768 si 32767 ture).

Iesirile numaratorului vor fi conectate la intrarile unor circuite tampon de tip 8286 prin care vor putea fi preluate, la o operatie de citire de la portul P0 (adresa **300h**) pe magistrala de date si transferate in microprocesor. Portul este configurat ca un port de 16 biti (cuvant), la o citire transferandu-se informatie pe toti cei 16 biti ai magistralei de date a sistemului. Deoarece bitii de date superiori (D10÷D15) nu contin informatie de pozitie provenind de la numarator, ei sunt conectati la masa pentru ca marimea citita de la port sa fie corect interpretabila in microprocesor, ca o marime pe 16 biti. De remarcat faptul ca circuitele 8286 au selectat doar sensul de transferare a datelor de la pinii A catre pinii B, prin conectarea pinului T la nivel 1 logic (la +5 V). Activarea circuitelor tampon este facuta la aparitia semnalului de selectie $\overline{P0}$, simultan cu semnalul de citire, $\overline{RD}$.

Asadar, prin program, aceasta interfata va fi utilizata astfel (port P0 $\equiv$ 300h):

IN AX, 300h ; dupa executie, registrul AX contine informatia de pozitie

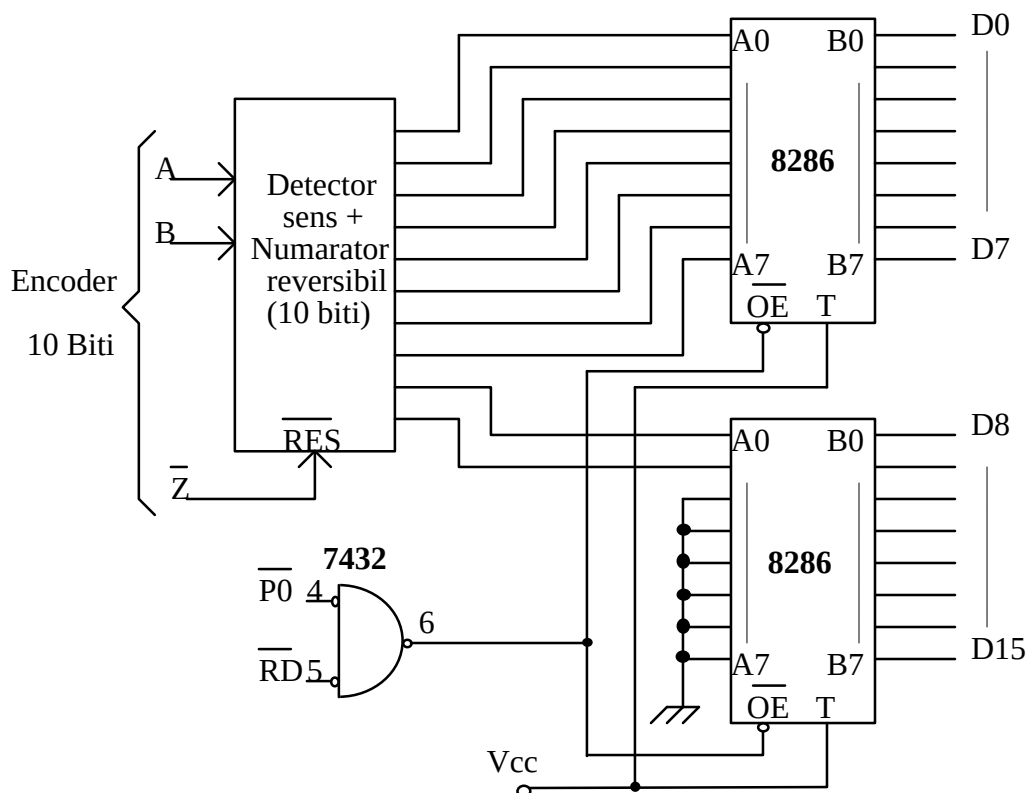


Figura 6.14. Schema interfetei traductorului de pozitie

6.3.3.2 Interfata traductorului de viteza

Traductorul de viteza utilizat în sistem — un tahogenerator de c.c. — va fi interfatat cu sistemul microprocesor prin intermediul unui convertor A/D de 10 biti, după cum s-a precizat în paragraful 6.1.9. Pentru aceasta, după cum se poate remarca din figura 6.15, s-a utilizat un convertor analog/numeric pe 10 biti de tipul **AD571**, produs de către firma Analog Devices. Deoarece mărimea aplicată la intrarea analogică **AIn** a circuitului AD571 poate avea o variație în plajă de $(-10 \div 10)$ V, semnalul furnizat de tahogenerator va trebui eventual divizat în prealabil printr-un divizor rezistiv, pentru a asigura respectarea acestei restricții, în cazul vitezei maxime a motorului — deci a tensiunii maxime furnizate de tahogenerator. Rezistențele R_1 și R_2 vor fi determinate din următoarele relații:

- curențul $U_{Tgmax}/(R_1+R_2)$ să fie cel puțin de 10 ori mai mare decât valoarea curențului absorbit de intrarea convertorului A/D.
- $R_2/(R_1+R_2) \leq 10/U_{tgmax}$, cu U_{Tgmax} măsurat în volți.

Pinul **AnCom** al convertorului se va conecta la masa părții analogice de măsură a schemei. Pinul **DigCom** se conectează la masa părții digitale a schemei. Pinul **B&C** trebuie utilizat pentru a genera începerea unei noi conversii (start conversie), la apariția unui puls pozitiv pe acest pin. După cum se remarca din figura, acest puls va fi generat, în schema propusă, la efectuarea unei operații de output la portul P1 al sistemului (adresă **308h**). Semnalul de selecție a portului, $\overline{P1}$, împreună cu semnalul de scriere, $\overline{WR}$, generează semnalul **ITGWR** aplicat pe pinul B&C al convertorului A/D. Bitii de date de ieșire ai convertorului (BIT1 la BIT10) vor fi conectați la intrările unor circuite tampon de tip 8286 prin care vor putea fi preluate, la o operație de citire de la portul P1 (adresă 308h) pe magistrala de date, și transferate în microprocesor. Portul este configurat ca un port de 16 biti (cuvant), la o citire transferându-se informație pe toți cei 16 biti ai magistralei de date a sistemului. De remarcat utilizarea bitului D15 de date ca bit indicator de stare a conversiei, prin conectarea acestui bit la ieșirea $\overline{DATA\ READY}$ a convertorului A/D. Astfel, pe durata unei conversii A/D, acest semnal are valoare logică 1, iar după încheierea conversiei, valoarea logică 0. Prin program, utilizatorul va putea efectua operații de citire de la portul P1 și, prin testarea bitului D15 de date, va putea determina starea conversiei A/D. [Ca o remarcă suplimentară, ceilalți biti de date (D10÷D14) nu sunt utilizați și ar putea, în principiu, să fie folosiți ca biti de intrare de uz general.] De remarcat faptul că circuitele 8286 au selectat doar sensul de transferare a datelor de la pinii A către pinii B, prin conectarea pinului T la nivel 1 logic (la +5 V). Activarea circuitelor tampon este făcută la apariția semnalului de selecție $\overline{P1}$, simultan cu semnalul de citire, $\overline{RD}$ (generând semnalul **ITGRD** aplicat intrărilor $\overline{OE}$ ale circuitelor 8286).

Asadar, prin program, aceasta interfata va fi utilizata astfel (P1 $\equiv$ 308h):

- OUT 308h, AX** ; determină startul conversiei A/D. Conținutul registrului
; AX nu este utilizat pentru această operație, la nivelul portului;
- IN AX, 308h** ; prin izolarea bitului D15, se poate detecta starea conversiei A/D
; (1 = conversie în curs de desfășurare; 0 = conversie încheiată);

IN AX, 308h ; prin izolarea bitilor D0÷D9 se obtine informatia de viteza,
 ; convertita numeric de catre convertorul A/D.
 ; ATENTIE! Aceasta informatie va fi valida doar daca bitul de stare
 ; a conversiei, conectat la bitul D15 de date, are valoarea logica 0.

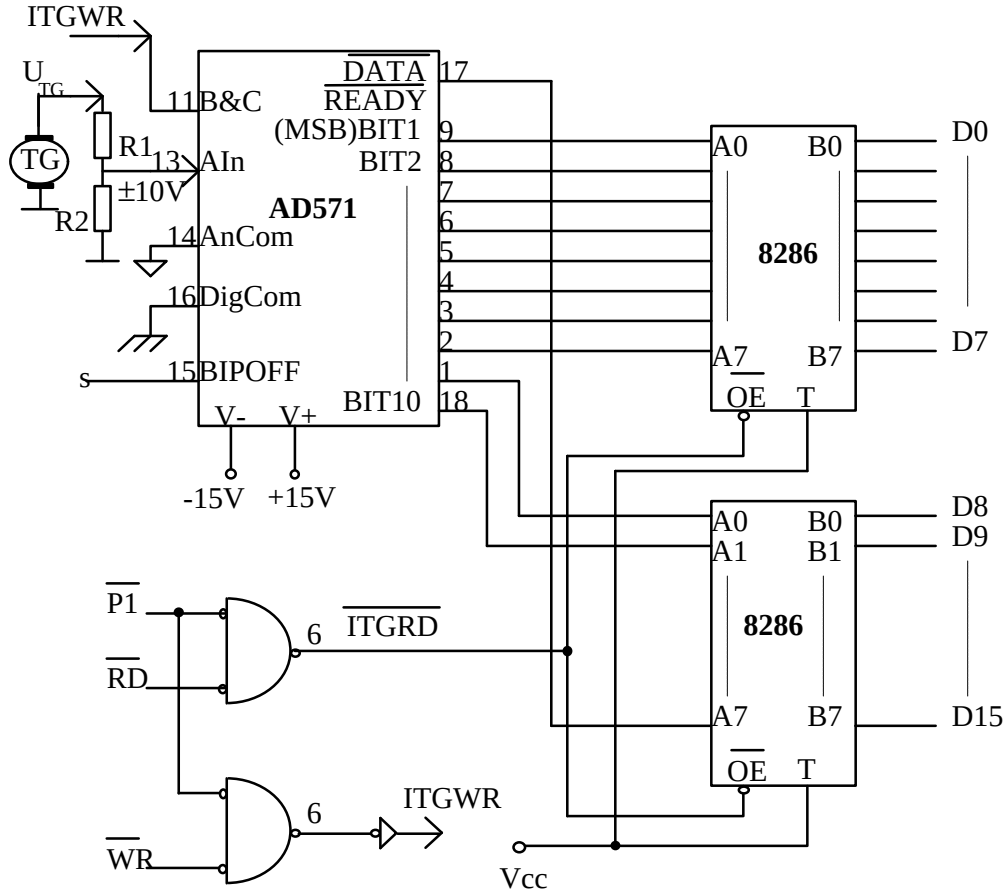


Figura 6.15. Schema interfetei traductorului de viteza

6.3.3.3 Interfetele pentru semnalizari si comenzi (I/E generale)

Dupa cum s-a mentionat, s-a prevazut pentru sistemul microprocesor un numar de biti de uz general pentru citirea de date numerice din proces (semnalizari diverse de la amplificator — elemente de diagnoza, contacte limitatoare de cursa, butoane de comanda manual/automata etc.) si pentru darea de comenzi numerice catre proces (comenzi catre amplificatorul de putere, comenzi pentru actionarea elementelor de tip binar — contactoare, frana electromagnetica etc.). Se poate considera ca un numar de 16 biti de intrare si 16 biti de iesire vor fi suficienti pentru aceste porturi de I/E de uz general. Asadar, sunt necesare un port de intrare si un port de iesire pentru acest scop. Figura 6.16 prezinta schema de implementare a acestor porturi, realizata cu:

- circuite de tip 8286 pentru portul de intrare (circuite tampon izoland semnalele de intrare de magistrala de date a sistemului). Sensul transferului de date prin aceste circuite fiind de

la intrările A către ieșirile B, pinul de control al sensului transferului de date, T, este conectat permanent la nivel logic "1". Activarea circuitelor tampon se face prin pinul $\overline{OE}$, utilizând semnalul de selecție a portului P2 (adresa **310h**), utilizat ca port de intrare—împreună cu semnalul $\overline{RD}$. Evident, se utilizează 2 circuite 8286, conectate la bitii inferiori D0÷D7, respectiv superiori D8÷D15, ai magistralei de date. Semnificația fiecăruia dintre semnalele SIG0 la SIG15, reprezentând cei 16 bitii de date de intrare ai portului, va fi definită în secțiunea de software a acestui capitol.

- circuite de tip 8282 pentru portul de ieșire, pentru memorarea unor comenzi date de microprocesor spre exterior. Ieșirile acestor circuite sunt în permanentă activate (pinul $\overline{OE}$ este conectat la masă), iar impulsul de memorare a datelor de pe magistrala de date este sintetizat pe baza semnalului de selecție a portului P2 (adresa **310h**), utilizat ca port de ieșire — împreună cu semnalul $\overline{WR}$. Sunt utilizate 2 circuite 8282 conectate la bitii inferiori D0÷D7, respectiv superiori D8÷D15, ai magistralei de date. Semnificația fiecăruia dintre semnalele CMD0 la CMD15, reprezentând cei 16 bitii de date de ieșire ai portului, va fi definită în secțiunea de software a acestui capitol.

Asadar, prin program, aceasta interfata va fi utilizata astfel (P2 $\equiv$ 310h):

OUT 310h, AX ; port ieșire; înscriere pe bitii de comanda CMD0÷CMD15 a
; conținutului registrului AX (bitii D0÷D15)

IN AX, 310h ; port intrare; citire în registrul AX (bitii D0÷D15) a
; semnalelor SIG0÷SIG15

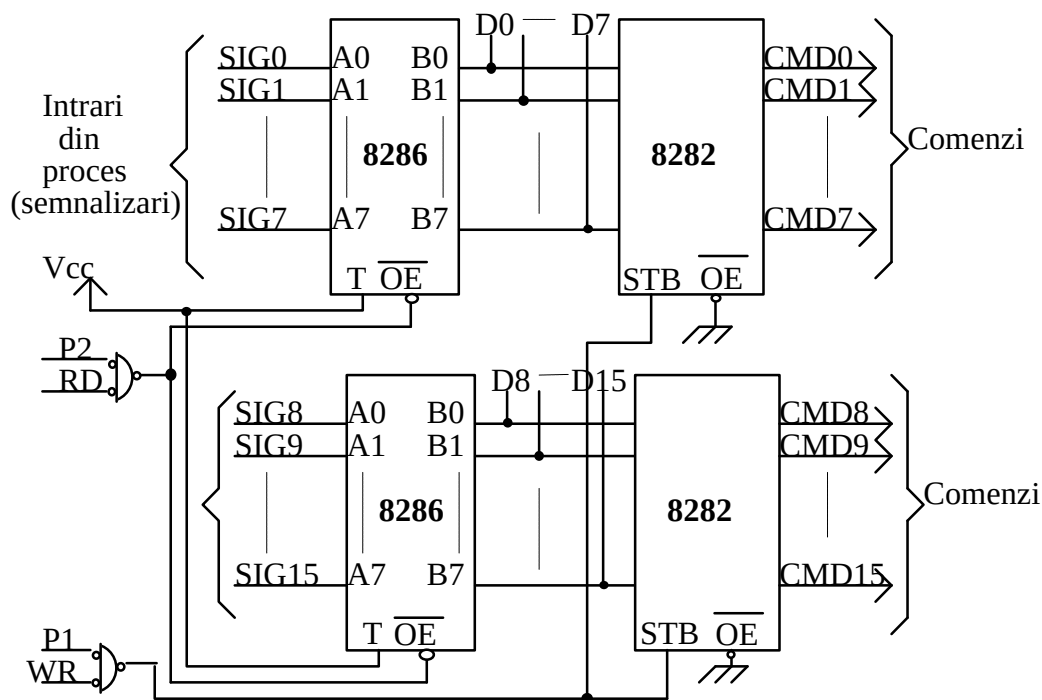


Figura 6.16. Schema interfetei porturilor de I/E de uz general (semnalizari și comenzi digitale)

6.3.3.4 Interfata pentru generarea marimii de comanda analogica

Pentru generarea semnalului analogic aplicat ca semnal de comanda a amplificatorului de forta al sistemului de pozitionare, este necesara o interfata ce contine un convertor digital/analog. Acesta va avea o rezolutie de 10 biti, conform celor prezentate in paragraful 6.1.8, la alegerea amplificatorului. Figura 6.17 prezinta schema de implementare a interfetei respective. Se considera ca interfata este configurata ca portul P0 de iesire (adresa 300h). Semnalul de selectie al acestui port, $\overline{P0}$, este utilizat impreuna cu semnalul de scriere, $\overline{WR}$, pentru a genera impulsul de memorare a datelor de pe magistrala de date a sistemului microprocesor, in circuitele de tip *latch* 8282 (prin conectare la pinul STB al acestor circuite). Iesirile acestor circuite, permanent activate (pinul $\overline{OE}$ conectat la masa), sunt aplicate — primii 10 biti, la intrarile de date ale unui convertor D/A de 10 biti, de tip AD7520, de productie *Analog Devices*. In ceea ce priveste conectarea celorlalti pini ai acestui circuit, de mentionat ca, deoarece iesirea convertorului este o iesire in curent, se utilizeaza un convertor curent-tensiune implementat cu ajutorul unui amplificator operational, dupa cum se poate vedea din schema prezentata. Pentru detalii asupra conectarii si utilizarii circuitului AD7520, a se consulta catalogul firmei *Analog Devices*. Iesirea analogica astfel obtinuta, denotata An.Out, va putea fi aplicata la intrarea de comanda a amplificatorului de curent al sistemului de pozitionare si va reprezenta valoarea impusa curentului prin motor, calculata in sistemul microprocesor, in bucla de control discret dupa stare. Ca o remarca suplimentara, deoarece bitii superiori de date (6 biti ai celui de-al doilea circuit 8282) nu sunt utilizati pentru convertorul D/A, in principiu, acesti biti ar putea fi folositi ca biti de uz general de iesire. Totodata, in cazul in care s-ar considera necesara utilizarea unui convertor D/A de rezolutie mai mare, se poate inlocui convertorul, fara a necesita modificari pe partea de interfata cu magistrala sistemului.

Asadar, prin program, aceasta interfata va fi utilizata astfel (P0 $\equiv$ 300h):

OUT 300h, AX ; port iesire; informatia din bitii D0÷D9 este convertita la
 ; nivelul convertorului D/A in informatie analogica

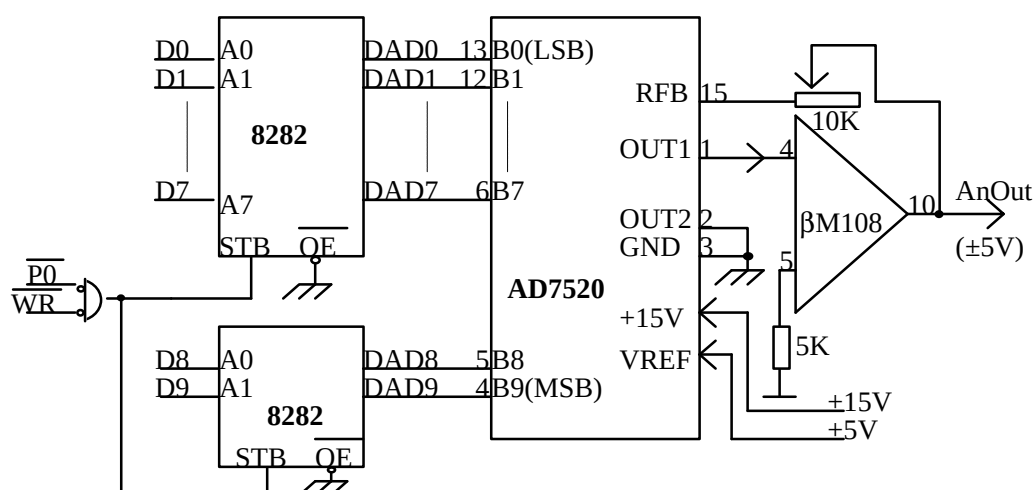


Figura 6.17. Schema interfetei pentru generarea marimii de comanda analogica

6.3.3.5 Interfata pentru tastatura

Consideram ca, pentru a permite utilizatorului sa genereze unele comenzi locale, este necesara utilizarea unei tastaturi si a unui afisaj cu structuri simple, dar suficiente pentru a ilustra principiul implementarii unor asemenea interfete. Pentru configuratii mai sofisticate, se vor utiliza circuite specializate (de exemplu circuitul 8257), sau chiar se va considera conectarea sistemului microprocesor la un monitor (tastatura-display), prin interfete specializate de tip serial sau module video dedicate.

Pentru tastatura, se va implementa o structura continand un numar de 32 de taste. Schema de implementare a interfetei este prezentata in figura 6.18. In esenta, se realizeaza o structura matriceala, avand 2 linii si 16 coloane, cele 32 de taste (denotate K_{00} la K_{0F} pe prima linie si K_{10} la K_{1F} pe a doua linie) fiind conectate in nodurile acestei retele matriceale. Coloanele sunt conectate prin niste rezistente la nivel logic "1", pe de o parte, si la magistrala de date a sistemului— prin intermediul unor circuite tampon, pe de alta parte. Selectia uneia dintre liniile structurii (cel mult una dintre cele doua linii de taste poate fi selectata la un moment dat) se face prin trecerea liniei respective la nivel logic "0". In consecinta, nivelul logic al semnalelor de pe cele 16 coloane va corespunde starii tastei de la intersectia coloanei respective cu linia selectata la acel moment: daca tasta nu este apasata, nivelul logic este "1", iar daca tasta este apasata, nivelul logic este "0". Daca se apasa o tasta de pe aceeaasi coloana, dar de pe o linie neselectata, nivelul logic de pe coloana respectiva nu este influentat.

Pentru a implementa structura propusa, sunt necesare 2 porturi de intrare. Bazat pe semnalul de selectie de port P3 (zona de adrese 318h÷31Fh), schema utilizeaza bitul de adresa A2 pentru a diferentia cele 2 porturi. Astfel, selectia celor doua linii de taste se va face generand semnalele de selectie de linie 0 - Sel.Linia0 (portul de intrare de la adresa 318h), respectiv de linie 1 - Sel.Linia1 (portul de intrare de la adresa 31Ch). Se utilizeaza semnalele $\overline{P3}$, $\overline{RD}$ si bitul de adresa A2, respectiv negatul sau, $\overline{A2}$. Concomitent cu generarea unuia din cele doua semnale de selectie de linie, se vor deschide circuitele tampon 8286 (prin activarea pinului $\overline{OE}$ al acestora) si nivelurile logice ale celor 16 coloane ale structurii matriceale a retelei de taste vor fi transmise pe magistrala de date a sistemului microprocesor. In consecinta, prin

instrucțiuni de citire de la porturile de la adresele 308h, respectiv 31Ch, se obține informația de poziție a primelor, respectiv a următoarelor 16 taste.

Asadar, prin program, aceasta interfata va fi utilizata astfel ($P3 \equiv 318h$):

IN AX, 318h ; selectează linia 0 de taste și transferă în registrul AX
; poziția acestora: 1 = tastă neapăsată; 0 = tastă apăsată

IN AX, 31Ch ; selectează linia 1 de taste și transferă în registrul AX
; poziția acestora: 1 = tastă neapăsată; 0 = tastă apăsată

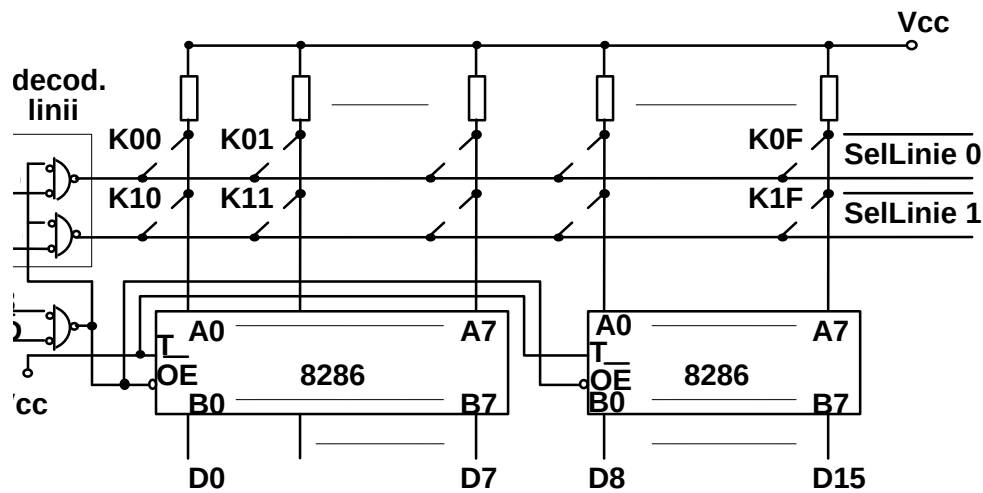


Figura 6.18. Schema interfeței pentru tastatură

6.3.3.6 Interfața pentru afișaj

Pentru a permite afișarea locală a unor informații privind funcționarea sistemului de poziționare, afișarea și, eventual, modificarea unor parametri ai schemei de control și a parametrilor de mișcare (referință), se prevede în schema, alături de tastatură prezentată în paragraful precedent, un afișaj realizat cu 4 cifre a câte 8 segmente fiecare. Figura 6.19 prezintă schema de implementare a acestei scheme. Se utilizează un număr de două porturi de ieșire configurate în zona de adrese ale portului P3 (318h÷31Fh). Bitul de adresă A2 este utilizat pentru a selecta cele două porturi la adresa 318h, respectiv 31Ch. Se utilizează semnalele $\overline{P3}$, $\overline{WR}$ și bitul de adresă A2, respectiv negatul, sau $\overline{A2}$. Cele două porturi sunt implementate utilizând câte două circuite de tip *latch* 8282. Semnalele de selecție a porturilor sunt aplicate pe pinii STB de înscriere a informației de pe magistrala de date a sistemului microprocesor în aceste circuite. La ieșirea circuitelor 8282 se conectează câte un element de afișaj cu segmente de tip LED. Cei 8 biți de date de ieșire sunt asociați celor 8 segmente ale unei cifre, în conformitate cu structura următoare:

B7				B0			
DP	g	f	e	d	c	b	a

S-au considerat segmentele afisajului cu anod comun, ceea ce implica conectarea acestora in schema conform figurii 6.19. Valoarea logica "1" pe o iesire a circuitelor 8282 implica segment stins, iar valoarea logica "0" implica segment aprins.

Rezistentele conectate in serie cu fiecare segment se dimensioneaza din urmatoarele considerente:

- caderea de tensiune pe un segment in starea aprins este de aproximativ 2 V;
- curentul prin segment trebuie sa fie suficient de mare pentru a asigura o luminozitate satisfacatoare a acestuia, uzual 7÷15 mA. Aceasta implica $R \leq (5-2) V / 0.007A \approx 430 \Omega$.
- curentul la iesirile circuitelor 8282 la nivel logic "0" nu trebuie sa depaseasca limita de catalog admisa, 30 mA, iar curentul prin segmente nu trebuie sa depaseasca limita admisa pentru acestea, 20 mA. Dimensionand R pentru a se asigura respectarea limitei minime de 20 mA, se obtine $R \geq (5-2)V/0.02A = 150 \Omega$.

Uzual, o valoare convenabila pentru R este 300 Ω .

In partea de software se vor include subrutine speciale de afisare a cifrelor si a unor caractere ce se pot configura pe afisaje de acest tip. Pentru fiecare cifra sau caracter afisabil, trebuie definita prin tabelare combinatia de biti corespunzand segmentelor cifrei care asigura aparitia simbolului cifrei sau a caracterului respectiv pe afisajul cu segmente.

Asadar, prin program, aceasta interfata va fi utilizata astfel (P3 $\equiv$ 318h):

OUT AX, 318h ; bitii D0÷D7 sunt utilizati pentru selectia segmentelor cifrei 0
; bitii D8÷D15 sunt utilizati pentru selectia segmentelor cifrei 1
; 1 = segment stins; 0 = segment aprins

OUT AX, 31Ch ; bitii D0÷D7 sunt utilizati pentru selectia segmentelor cifrei 2
; bitii D8÷D15 sunt utilizati pentru selectia segmentelor cifrei 3
; 1 = segment stins; 0 = segment aprins

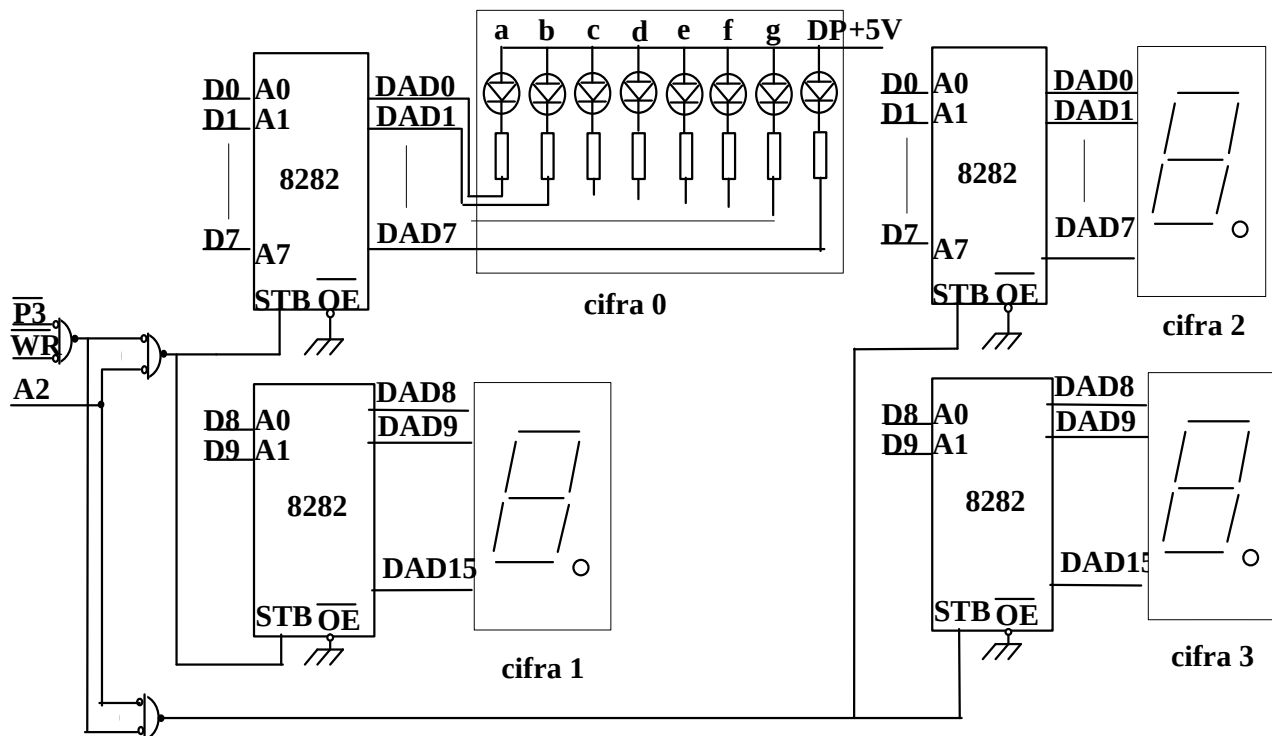


Figura 6.19. Schema interfetei pentru afisaj

6.3.3.7 Interfata ceasului de timp real

Pentru a putea implementa o structura de control discret, este necesara executia procedurii de control la intervale de timp egale cu pasul de esantionare al procedurii de control (1 ms, in cazul aplicatiei studiate). In acest sens, este necesara utilizarea unui mod de masurare a acestor intervale de timp, operatie efectuata in mod uzual prin utilizarea unui ceas de timp real care, la sfarsitul fiecarui pas de esantionare, este utilizat pentru a furniza un semnal de intrerupere catre microprocesor. O posibilitate ar fi de a utiliza un circuit de numarare cu structura data, furnizand un semnal de frecventa fixa. Pentru a avea insa o schema flexibila, este preferabila utilizarea unui circuit de numarare programabil. Acesta va furniza semnale periodice cu frecventa programabila intr-o anumita gama de valori, ceea ce permite modificarea usoara a perioadei de esantionare, prin simpla modificare a secventei de programare a circuitului respectiv.

Pentru sistemul studiat, se propune utilizarea unui circuit de ceas programabil de tipul INTEL 8253, circuit care contine un numar de 3 numaratoare de cate 16 biti fiecare, programabile in mai multe moduri, putand deci diviza un semnal de ceas de intrare aplicat fiecaruia dintre cele 3 numaratoare, cu o valoare numerica cuprinsa intre 2 si 65536. Figura 6.20 prezinta structura interfetei realizate in acest scop. Circuitul 8253 utilizeaza intern un numar de 4 porturi, astfel ca s-a utilizat pentru selectia circuitului semnalul de selectie de port $\overline{P0}$ ca semnal de selectie de cip, $\overline{CS}$. Bitii de adresa A1 si A2 au fost conectati direct pe cip, respectiv la semnalele A0 si A1 ale acestuia. Semnalul C4M de la iesirea circuitului 8284 de ceas al sistemului este

divizat într-un numărător de tip 7493, pentru a fi aplicat la intrarea de ceas CLK0 a circuitului 8253, deoarece ca data de catalog acest circuit accepta un semnal de frecvență maximă 2 MHz pentru acest semnal. Prin programarea circuitului, frecvența semnalului de ieșire CLKTIM0 de la pinul OUT0 al circuitului 8253 poate fi deci situată în gama de minim 31 Hz și maxim 1 MHz, deci cu perioada cuprinsă între 1 μ s și aproximativ 30 ms. Validarea permanentă a numărării este asigurată prin conectarea semnalului GATE0 la nivel logic "1". Celelalte două numărătoare ale circuitului 8253 sunt neutilizate în structura propusă, rămânând ca resurse ale sistemului utilizabile ulterior. Pentru detalii referitoare la structura internă și la modul de programare a circuitului 8253, a se consulta documentația de firmă referitoare la acest circuit.

Asadar, prin program, aceasta interfata va fi utilizata astfel ($P4 \equiv 320h$):

- la adresa 320h: port date / stare contor 0
- la adresa 322h: port date / stare contor 1
- la adresa 324h: port date / stare contor 2
- la adresa 326h: port control circuit 8253

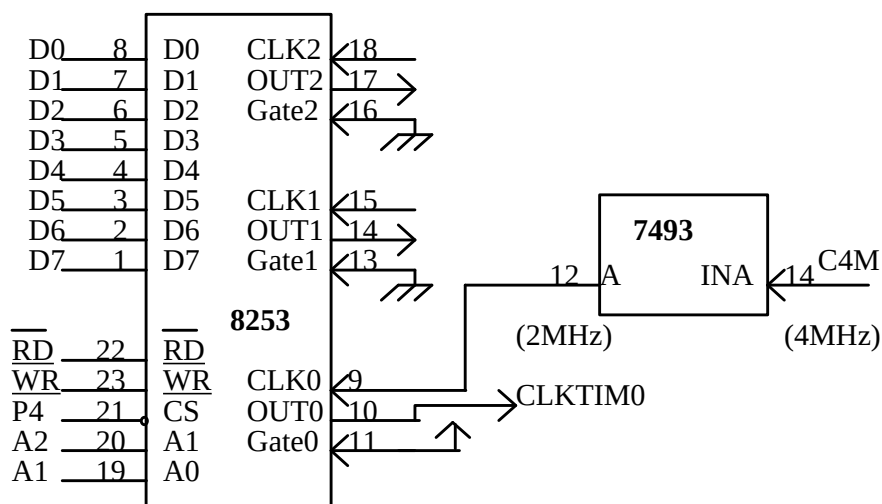


Figura 6.20. Schema interfeței pentru ceasul de timp real

6.3.3.8 Interfața controlerului de întreruperi

Pentru a gestiona întreruperile externe microprocesorului prin semnalul INTR, este necesară utilizarea unui controler de întreruperi de tip 8259A (vezi cap. 4, pentru detalii referitoare la structura, funcțiile și modul de programare și utilizare ale acestui circuit). Figura 6.21 prezintă schema de conexiuni a unui asemenea circuit utilizat în sistemul microprocesor propus. Bitii de date ai circuitului se leagă la bitii de date D0-D7 ai magistralei de date, selecția de cip este asigurată de semnalul de selecție a portului $\overline{P5}$ și, din motive didactice, se presupune că semnalul furnizat de circuitul de ceas de timp real 8253, CLKTIM0 (v. par. 6.3.3.9) este aplicat la intrarea IR3 a circuitului 8259A, pentru a furniza întreruperea de timp real la terminarea fiecărei perioade de esanționare și a determina apelarea rutinei de control discret al

sistemului de pozitionare. Sectiunea de software va prezenta modul de utilizare prin program a acestei structuri. Circuitul 8259A functioneaza ca circuit master unic (se utilizeaza un singur controler de intreruperi), deci pinul $\overline{SP/EN}$ este conectat la nivel logic "1", iar pinii $CAS0-CAS2$ sunt neutilizati. Iesirea $\overline{INT}$ a circuitului se conecteaza la intrarea INTR a microprocesorului 8086, iar intrarea $\overline{INTA}$ la iesirea cu acelasi nume a microprocesorului. Bitul de adresa A1 este utilizat la nivelul cipului 8259A, pe intrarea A0 a acestuia. Tot din motive didactice, se presupune ca semnalul care indica functionarea corecta a amplificatorului, $\overline{OK}$ convertizor, este conectat la pinul de intrerupere nemascabila a microprocesorului, NMI.

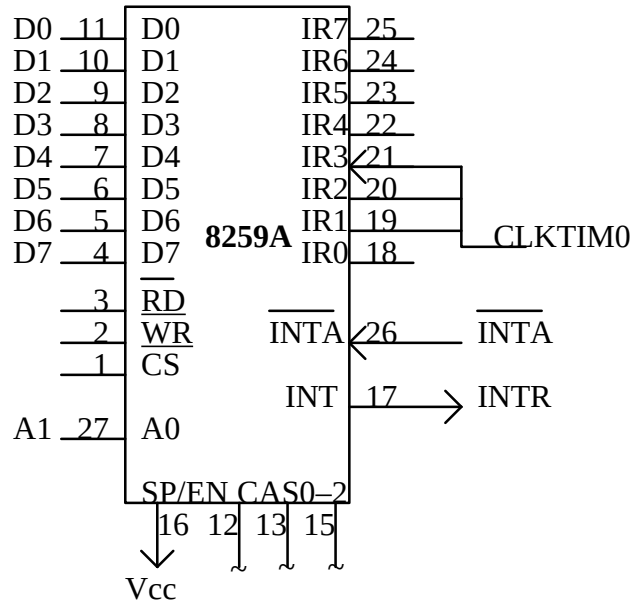


Figura 6.21. Schema interfetei pentru controlerul de intreruperi

Asadar, prin program, aceasta interfata va fi utilizata astfel ($P5 \equiv 328h$):

- la adresa 328h: port cuvinte initializare 8259A
- la adresa 32Ch: port cuvinte de operare 8259A

(vezi documentatia de firma referitoare la circuitul 8259A, pentru detalii).

6.4 PROIECTAREA STRUCTURII SOFTWARE A SISTEMULUI

In paragraful 6.3 s-a realizat proiectarea structurii hardware a sistemului microprocesor care implementeaza controlul discret al actionarii electrice de c.c. S-au configurat circuitele, s-au definit zonele de adrese si modul de interconectare al tuturor componentelor sistemului microprocesor. In continuare, se poate trece la specificarea unor elemente suplimentare legate de implementarea programului microprocesorului, urmata de elaborarea programului propriu-zis.

Trebuie precizat de la inceput ca programul ce va fi prezentat in continuare este o versiune minimala, necesara pentru a implementa functiile esentiale ale aplicatiei studiate. Bazate pe aceeasi structura hardware, se pot imagina programe mai complexe, cu functii si facilitati suplimentare.

6.4.1 Specificatiile de programare ale resurselor sistemului

În continuare, reluăm specificațiile fiecăreia dintre resursele sistemului, importante din punct de vedere al programului: adrese, semnificația bitilor interfețelor sistemului etc.

• Memoria PROM

Ocupă zona de adrese (**0FC000h÷0FFFFh**). La adresa **0FFFF0h** trebuie plasată prima instrucțiune din program (executată după inițializarea microprocesorului, prin aplicarea semnalului de RESET). Programul propriu-zis va fi plasat de la startul memoriei PROM, de la adresa **0FC000h**. Memoria PROM va fi structurată ca în figura 6.22.

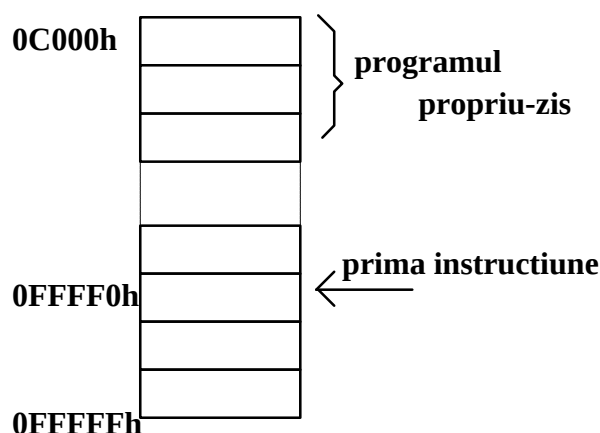


Figura 6. 22. Organizarea memoriei PROM

• Memoria RAM

Ocupă zona de adrese (**0h÷0FFFh**). La adresa **8h** se va plasa vectorul de întrerupere pentru tratarea întreruperii nemascabile NMI. Considerăm ca adresa de bază pentru vectorii de întrerupere externă este **80h**. Această adresă va fi utilizată la programarea circuitului controler de întreruperi 8259A. Întreruperea de la ceasul de timp real fiind conectată la intrarea IR3 a acestui circuit, adresa la care trebuie plasat vectorul de întrerupere asociat acestui nivel de întrerupere va fi **8Ch**. Variabilele generale ale programului vor fi amplasate începând cu adresa **100h**. Stiva programului va fi amplasată la sfârșitul zonei de memorie RAM, de la adresa **0FFFh** înspre adrese mici.

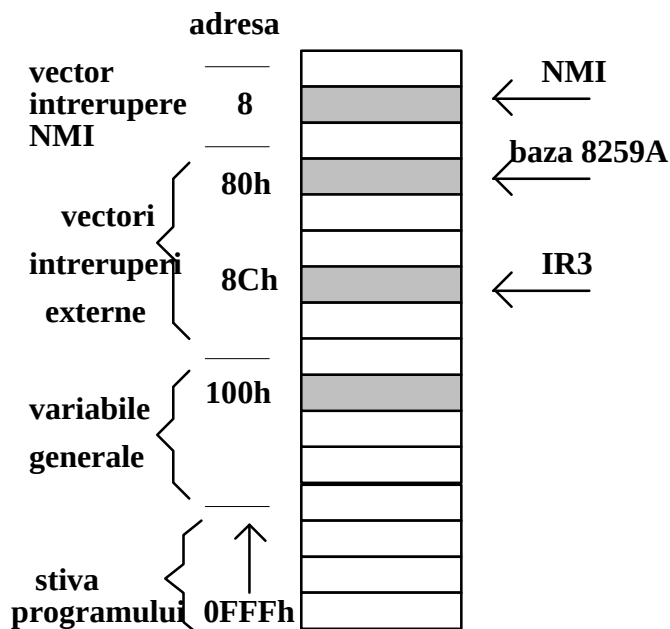


Figura 6.23. Structura memoriei RAM

- Interfata traductorului de pozitie

Port de intrare, la adresa **300h**. Informatia de pozitie este citita pe bitii (**D0÷D9**) de date.

- Interfata traductorului de viteza

Port de iesire, la adresa **308h**. Start conversie A/D. Bitii de date neutilizati.

Port de intrare, la adresa **308h** :

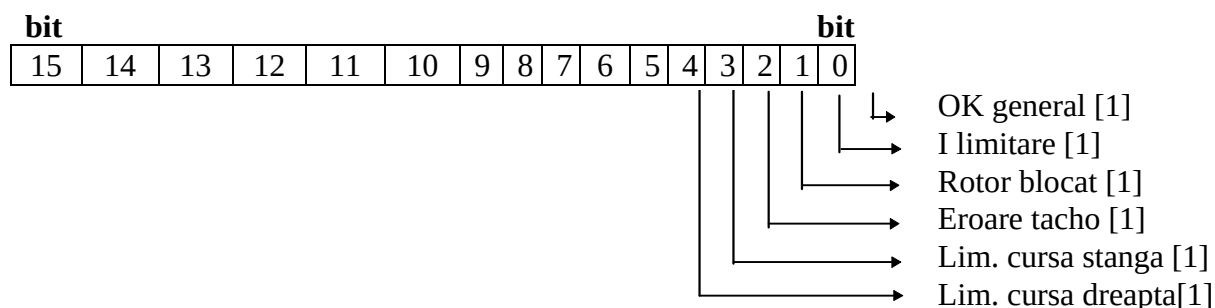
Bitul **D15** indica starea conversiei : **D15=1** conversie neterminata

D15 = 0 conversie terminata

Bitii **D0÷D9** contin data convertita, daca conversia s-a incheiat (**D15=0**)

- Semnalizari generale

Port de intrare, la adresa **310h**. Se presupune urmatoarea semnificatie a bitilor de date:

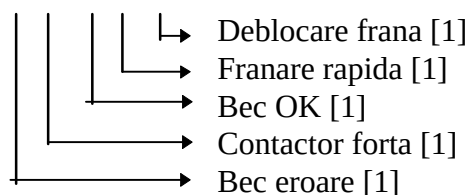


Intre paranteze s-a indicat valoarea logica pentru care semnalul este activ. Se remarca faptul ca bitii de date **D6÷D15** nu sunt utilizati in structura actuala, ei putand fi conectati la alte semnalizari, definite ulterior, si tratati in mod corespunzator in program.

- Comenzi generale

Port de iesire, la adresa **310h**. Se considera urmatoarea semnificatie a bitilor de date:

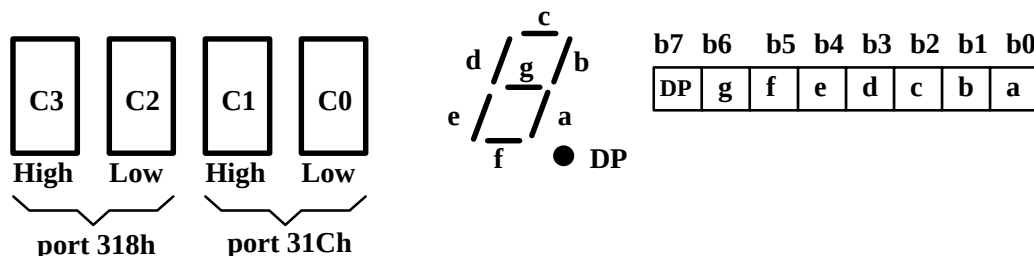
bit															bit
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0



Intre paranteze s-a indicat valoarea logica pentru care semnalul este activ. Se remarca faptul ca bitii de date **D5÷D15** nu sunt utilizati in structura actuala, ei putand fi conectati la alte comenzi, definite ulterior, si tratati in mod corespunzator in program.

• Afisaj cu 4 cifre

2 porturi de iesire la adresele **318h** si **31Ch**. Se considera urmatoarea structurare a informatiei:

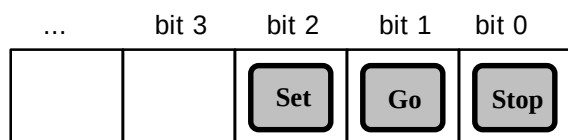


Cele 4 cifre sunt grupate cate doua pentru fiecare port, iar bitii asociati fiecarei cifre sunt corelati, conform figurii prezentate, cu segmentele acesteia. Dupa cum s-a mentionat deja, valoarea 0 a unui bit va implica aprinderea segmentului asociat acestuia.

• Tastatura

Port de intrare, la adresa de baza **318h**. Se considera urmatoarea semnificatie a bitilor de date:

- **linia 0 de taste** (taste de comanda) (port intrare, adresa **318h**):



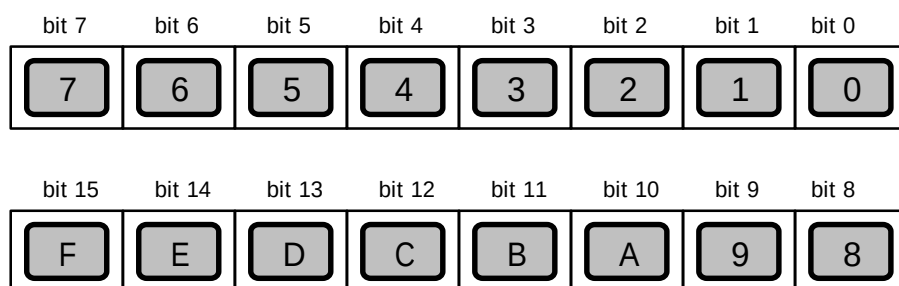
Tasta **Set** va fi utilizata pentru a seta pozitia finala a miscarii.

Tasta **Go** va fi utilizata pentru a comanda inceperea miscarii

Tasta **Stop** va fi utilizata pentru a opri miscarea

Celelalte taste ale liniei 0 nu sunt utilizate in structura actuala a programului, deci pot primi semnificatia dorita pentru dezvoltari ulterioare si adaugari de functii noi.

- **linia 1 de taste** (taste numerice, numerotare hexazecimala) (port intrare, adresa **31Ch**):



• Comanda D/A

Port de iesire, la adresa **300h**. Bitii **D0÷D9** trebuie sa contina informatia, in cod complementar fata de 2, a marimii digitale a comenzii (bit **D9** $\equiv$ bit de semn).

• Porturi specializate

Se utilizeaza un controler de intreruperi 8259A, cu adresa de baza 320h (porturi localizate doar la adrese pare) si un ceas de timp real 8253, cu adresa de baza 328h (porturi localizate doar la adrese pare).

Considerand perioada de esantionare $T = 1\text{ms}$, rezulta cuanta de divizare a ceasului de timp real (frecventa de intrare, 2 MHz) de $N = 2000$.

6.4.2 Organizarea programului

Dupa cum s-a mai mentionat deja, programul va avea o structura de baza simplificata. In esenta, s-a urmarit implementarea tuturor caracteristicilor fundamentale necesare pentru a se putea efectua initializarile de baza ale componentelor hardware si software, pentru a se accepta comenzi elementare de la tastatura sistemului, a se afisa pozitia curenta a motorului si a se implementa algoritmul de reglare discreta dupa stare a pozitiei.

In figura 6.24 este prezentata schema logica a programului principal, iar in figura 6.25 schema logica a subrutinei de tratare a intreruperii de timp real in care se implementeaza schema de control discret propusa si a rutinei de tratare a intreruperii nemascabile NMI.

6.4.3 Prezentarea programului implementat

In continuare, se va prezenta listingul programului complet. Se utilizeaza conventiile, comenzile si directivele acceptate de catre programul de asamblare ASM-86 al firmei INTEL,

realizat pentru microprocesorul 8086. A se consulta documentatia de firma, pentru detalii suplimentare referitoare la aceste programe.

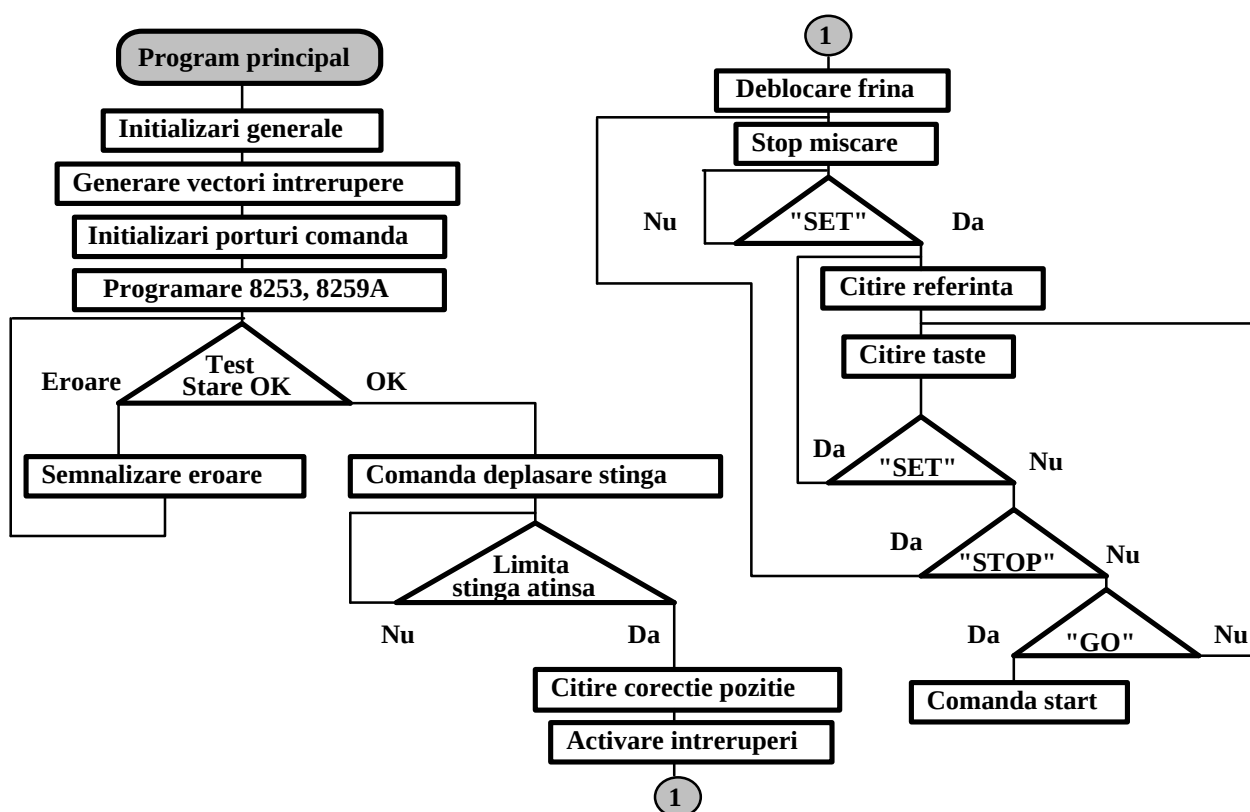


Fig. 6.25. Organigrama programului principal

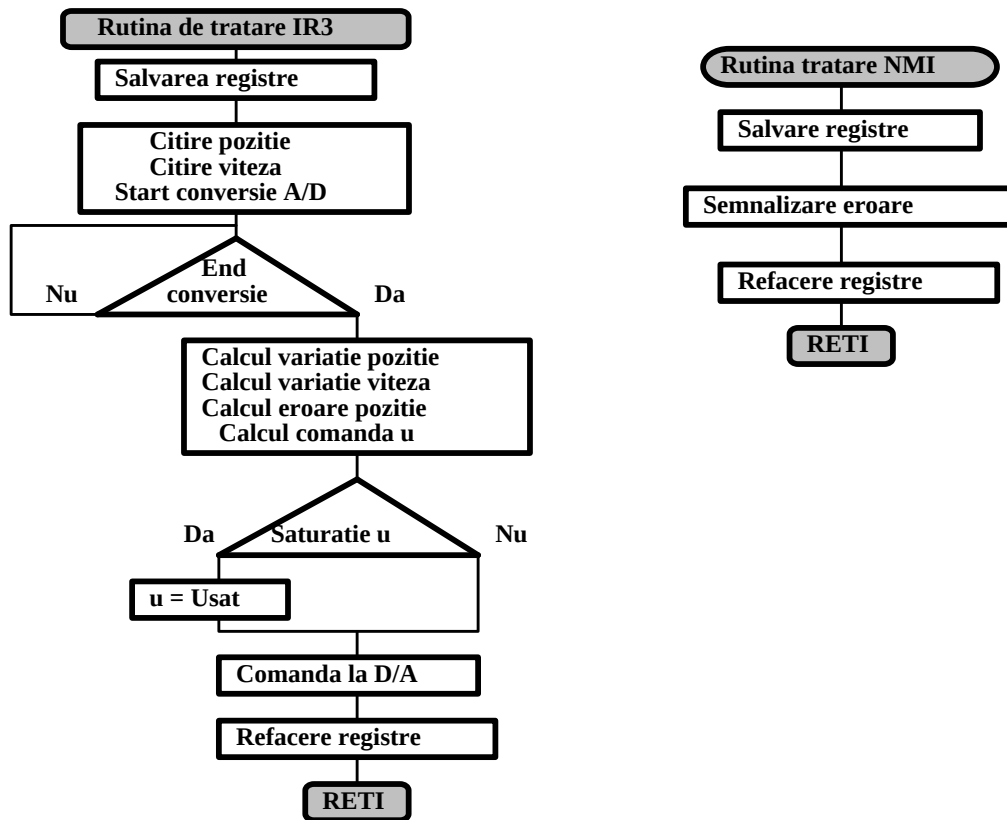


Fig. 6.26. Organigrama subrutinelor de tratare a intreruperilor

Program pentru controlul unui sistem de pozitionare cu motor de c.c.

(A) Definirea variabilelor generale ale programului

porturi generale:

PPOZITIA	EQU	300H	; port interfata pozitie (I)
PADSTART	EQU	308H	; port start conversie A/D (O)
PADDATA	EQU	308H	; port citire stare/date conversie A/D (I)
PSEMNAL	EQU	310H	; port citire semnalizari (I)
PCMD	EQU	310H	; port dare comenzi digitale (O)
PDADATA	EQU	300H	; port convertor D/A (O)
PKYBD0	EQU	318H	; port 0 tastatura (citire taste comenzi) (I)
PKYBD1	EQU	31CH	; port 1 tastatura (citire taste numerice) (I)
PDISP0	EQU	318H	; port 0 afisaj (cifre: C0 - low, C1 - high) (O)
PDISP1	EQU	31CH	; port 1 afisaj (cifre: C2 - low, C3 - high) (O)

ceas timp real 8253:

PRTC0	EQU	320H	; port timer 0 RTC (I/O)
PRTC1	EQU	322H	; port timer 1 RTC (I/O)
PRTC2	EQU	324H	; port timer 2 RTC (I/O)
PRTCCMD	EQU	326H	; port control RTC (O)

controler intreruperi 8259A:

PICICW1	EQU	328H	; port ICW1 PIC (O)
PICICW24	EQU	32AH	; port ICW2-ICW4 PIC (O)
PICOCW1	EQU	32AH	; port OCW1 PIC (O)

```

PICOCW23    EQU    328H    ; port OCW2-OCW3 PIC (O)
;           adrese vectori intrerupere
INTVNMI     EQU    8H      ; adresa vector intrerupere NMI
INTVRTC0    EQU    8CH     ; adresa vector intrerupere IQ3
;
;-----
;           Variabile generale (localizate in memoria RAM)
;-----
ORG 100H      ; zona de date generale incepe de la adresa 100H
;-----
POZCOR      DW      0       ; corectie pozitie [1 word; initial = 0]
POZREF      DW      0       ; referinta pozitie ( $\Theta^*$ ) [1 word; initial = 0]
POZCRT      DW      0       ; pozitia curenta ( $\Theta_i$ ) [1 word; initial = 0]
VITCRT      DW      0       ; viteza curenta ( $\Omega_i$ ) [1 word; initial = 0]
VIM1        DW      0       ; viteza la pasul anterior ( $\Omega_{i-1}$ ) [1 word; initial = 0]
UCRT        DW      0       ; comanda curenta ( $u_i$ ) [1 word; initial = 0]
DPOZ        DW      0       ; variatia pozitiei ( $\Delta\Theta = \Theta_i - \Theta_{i-1}$ ) [1 word; initial = 0]
DVIT        DW      0       ; variatia vitezei ( $\Delta\Omega = \Omega_i - \Omega_{i-1}$ ) [1 word; initial = 0]
POZIM1      DW      0       ; pozitia la pasul anterior ( $\Theta_{i-1}$ ) [1 word; initial = 0]
VITIM1      DW      0       ; viteza la pasul anterior ( $\Omega_{i-1}$ ) [1 word; initial = 0]
PERROR      DW      0       ; eroarea de pozitie ( $\epsilon = \Theta_i^* - \Theta_i$ ) [1 word; initial = 0]
UIM1        DW      0       ; comanda la pasul anterior ( $u_{i-1}$ ) [1 word; initial = 0]
;
;=====
;

```

```

; (B) Zona de cod a programului (localizata in memoria PROM)
; -----
;
; Programul principal
; -----
;
; ORG 0FFFF0H ; adresa de salt la RESET
; -----
; JMP MAIN ; salt la zona de program propriu-zis
;
; ORG 0FC000H ; adresa de inceput a memoriei PROM
; -----
;
MAIN: ; inceput program principal propriu-zis
MOV CS, 0FC00H ; initializare CS, adresa de inceput PROM
MOV DS, 0 ; initializare DS, adresa de inceput RAM
MOV SS, 0 ; initializare SS, adresa de inceput RAM
MOV SP, 1000H ; initializare SP, adresa de sfarsit RAM
;
; scriere vectori de intrerupere in memoria RAM
LEA AX, NMIERR ; AX = adresa rutinei de tratare a NMI
MOV INTVNMI, AX ; memorare vector tratare NMI in RAM
LEA AX, SRTC0 ; AX = adresa rutinei de tratare a IR3
MOV INTVRTC0, AX ; memorare vector tratare IR3 in RAM
;
; initializari porturi
MOV AX, 0
OUT PDADATA, AX ; comanda D/A = 0
OUT PCMD, AX ; comenzi generale = 0
;
; programare 8253
MOV AL, 06H ; set control 8253
OUT PRTCMD, AL ; set mod 3, timer 0, numarare binara
MOV BX, 2000 ; BX = N, cuanta de divizare
MOV AL, BL
OUT PRTC0, AL ; set low(N)
MOV AL, BH
OUT PRTC0, AL ; set high(N)
;
; programare 8259A
MOV AL, 00010111B ; set ICW1:
OUT PICICW1, AL ; edge / call 4/ single / +ICW4
MOV AL, 10000000B ; set ICW2:
OUT PICICW24, AL ; adresa de baza a vectorilor de intrerupere, 80H
MOV AL, 00000001B ; set ICW4:
OUT PICICW24, AL ; mod 8086, normal EOI
MOV AL, 11110111B ; set OCW1:
OUT PICOCW1, AL ; nemascat doar IR3
; -----
; alimentare partea de forta
MOV AX, 01000B
OUT PCMD, AX ; actionare contactor de forta
MOV AX, 2000
CALL WAIT1MS ; asteptare 2 secunde

```

```

;
; test daca convertizorul este OK
TESTOK:
    IN     AX,    PSEMNAL
    AND    AX,    1          ; izolare bit de OK (B0)
    JZ     ERROK           ; eroare OK, se semnalizeaza pe bec eroare
;
; deplasare pentru gasire pozitie initiala
    MOV    AX,    1101B      ; comenzi:
    OUT    PCMD, AX          ; deblocare frana / alimentare forta / aprindere bec OK
    MOV    AX,    -25        ; referinta mica de viteza negativa
    OUT    PDADATA, AX      ; deplasare stanga cu viteza mica
;
LOOP0:
    IN     AX,    PSEMNAL    ; citire semnalizari
    MOV    BX,    AX        ; salvare in BX
    AND    AX,    1          ; test OK (izolare bit B0)
    JZ     ERROK           ; salt, daca nu e OK
    MOV    AX,    BX
    AND    AX,    100B       ; test bit limitator cursa stanga
    JZ     LOOP0           ; reia, daca nu s-a atins limita stanga
;
; limita stanga s-a atins
    IN     AX,    POZITIA    ; citire pozitia la limita stanga
    MOV    POZCOR, AX       ; memorare ca si corectie de pozitie
;
    MOV    AX,    0
    MOV    POZREF, AX       ; initial, referinta pozitie = 0
    MOV    POZCRT, AX       ; initial, pozitia curenta = 0
    MOV    UIM1, AX         ; set ui-1 = 0
    MOV    POZIM1, AX       ; set  $\Theta$ l-1 = 0
    MOV    VITIM1, AX       ; set  $\Omega$ l-1 = 0
;
; activare intreruperi
    STI
;
STOPCMD:
    MOV    AX,    1111B      ; setare comenzi:
    OUT    PCMD, AX          ; deblocare frana / franare rapida / bec OK /
                                ; contactor forta alimentat
    MOV    AX,    POZCRT
    CALL   AFISAJ           ; afisare pozitia curenta
;
BIGLOOP:
    CALL   KEYBDC           ; citire comanda taste in AL
    AND    AL,    1          ; test, daca comanda "SET"
    JNZ    BIGLOOP         ; ignora alte taste, reia test
;
; s-a apasat tasta "SET", se citeste pozitia de referinta
BEGCIT:
    CALL   KEYBDD           ; citire o cifra hexa, transmisa in AL
    MOV    BX,    AL        ; salvare in BX
    CALL   AFISAJ           ; afisare noua valoare
; citire cifra a doua
    CALL   KEYBDD

```

```

MOV CL, 4
ROL BX, CL ; rotire BX cu 4 biti la stanga
OR BL, AL ; cifra 2 pe ultimii 4 biti ai BL
MOV AX, BX
CALL AFISAJ ; afisare valoarea curenta a referintei
; citire cifra a treia
CALL KEYBDD
MOV CL, 4
ROL BX, CL ; rotire BX cu 4 biti la stanga
OR BL, AL ; cifra 3 pe ultimii 4 biti ai BL
MOV AX, BX
CALL AFISAJ ; afisare valoarea curenta a referintei
; citire cifra a patra
CALL KEYBDD
MOV CL, 4
ROL BX, CL ; rotire BX cu 4 biti la stanga
OR BL, AL ; cifra 4 pe ultimii 4 biti ai BL
MOV AX, BX
CALL AFISAJ ; afisare valoarea curenta a referintei
; BX contine cele 4 cifre hexa pe cei 16 biti ai sai
;
WAITCMD:
CALL KEYBDC ; citire comanda tastatura
MOV CL, AL ; salvare cod comanda
AND AL, 1 ; test, daca "SET"
JZ BEGCIT ; daca "SET", reia citirea referintei
;
MOV AL, CL
AND AL, 100B ; test, daca "STOP"
JZ STOPCMD ; salt, daca "STOP"
;
MOV AL, CL
AND AL, 10B ; test, daca "GO"
JNZ WAITCMD ; salt; daca nu, reia asteptare comanda
;
; s-a dat "GO", setare noua referinta
MOV POZREF, BX
MOV AX, 1101B ; setare comenzi:
OUT PCMD, AX ; deblocare frana rapida
JMP WAITCMD ; reia asteptare comanda
; In acest timp, bucla de control lucreaza pentru
; a deplasa motorul la noua pozitie de referinta
;
;-----
EROKK:
; Secventa apelata, daca convertizor defect
MOV AX, 11000B ; setare comenzi:
OUT PCMD, AX ; forta alimentata / frana blocata /
; bec OK - stins / bec eroare - aprins
MOV AX, 0E111H ; afisare cod eroare OK, "E111"
CALL AFISAJ
JMP TESTOK ; reia test convertizor
;-----
;

```

```

;-----
; Subrutine diverse
;-----
WAIT1MS:
; Rutina de asteptare multiplu de 1 ms. Implementare software
; (Neimplementata ...)
; Argumente: <I>: AX = N ms de asteptat
;
; ...
;
; RET
;-----
AFISAJ:
; Rutina de afisare pe display a 4 cifre hexazecimale
; Argumente: <I>: AX = cele 4 cifre de afisat
;           Bit 15 ... 12 11 ... 8 7 ... 4 3 ... 0
;           | C3 | C2 | C1 | C0 |
; Registre afectate: AX, BX, CX, DX
;
MOV    BX,    AX           ; BX = numarul initial
;
; cifra 0
LEA    CX,    TABELH      ; CX = adresa tabelului de coduri cifre
AND    AX,    0FH         ; izolare cifra 0 (ultimii 4 biti)
ADD    CX,    AX          ; CX = adresa codului cifrei, din tabel
MOV    DL,    [CX]        ; citire cod cifra 0, memorare in DL
;
; cifra 1
MOV    AX,    BX          ; AX = numarul initial
AND    AX,    0F0H        ; izolare cifra 1 (bitii 4÷7)
MOV    CL,    4           ;
ROR    AX,    CL          ; rotire AX cu 4 biti spre dreapta
; cifra 1 pe bitii B0÷B3 ai AL
LEA    CX,    TABELH      ; CX = adresa tabelului de coduri cifre
ADD    CX,    AX          ; CX = adresa codului cifrei, din tabel
MOV    DH,    [CX]        ; citire cod cifra 1, memorare in DH
;
MOV    AX,    DX          ; cod cifre 1 si 0 (AH, AL)
OUT    PDISP0,    AX      ; afisare cifrele 1 si 0 pe display
;
; cifra 2
MOV    AX,    BX          ; AX = numarul initial
AND    AX,    0F00H       ; izolare cifra 2 (bitii 8÷11)
MOV    CL,    8           ;
ROR    AX,    CL          ; rotire AX cu 8 biti spre dreapta
; cifra 2 pe bitii B0÷B3 ai AL
LEA    CX,    TABELH      ; CX = adresa tabelului de coduri cifre
ADD    CX,    AX          ; CX = adresa codului cifrei, din tabel
MOV    DL,    [CX]        ; citire cod cifra 2, memorare in DL
;
; cifra 3
MOV    AX,    BX          ; AX = numarul initial
AND    AX,    0F000H      ; izolare cifra 3 (bitii 12÷15)
MOV    CL,    12          ;
ROR    AX,    CL          ; rotire AX cu 12 biti spre dreapta
; cifra 3 pe bitii B0÷B3 ai AL

```

```

    LEA    CX,    TABELH    ; CX = adresa tabelului de coduri cifre
    ADD    CX,    AX        ; CX = adresa codului cifrei, din tabel
    MOV    DH,    [CX]      ; citire cod cifra 3, memorare in DH
;
    MOV    AX,    DX        ; cod cifre 3 si 2 (AH, AL)
    OUT    PDISP1,    AX    ; afisare cifrele 3 si 2 pe display
;
    RET
;-----
KEYBDC:
; Rutina de citire a tastelor de comanda
; Argumente: <O> : returneaza codul de comanda in AL
;
    IN     AL,     PKYBD0    ; citire port taste comenzi
    CMP    AX,     0FFFFH    ; test, daca tasta apasata
    JZ     KEYBDC           ; daca nici o tasta nu e apasata, reia
;
    RET                    ; daca s-a apasat o tasta, return
                        ; AL = data citita (tasta apasata seteaza
                        ; bitul corespunzator pe 0)
;-----
KEYBDD:
; Rutina de citire a tastelor numerice de la "0" la "F"
; Argumente: <O>: returneaza in AL codul hexazecimal al tastei citite
; (valoare intre 0 si F)
;
    IN     AX,     PKYBD1    ; citire de la portul de taste numerice
    CMP    AX,     0FFFFH
    JZ     KEYBDD           ; reia, daca nu s-a apasat tasta
; s-a apasat o tasta numerica
    MOV    CX,     AX        ; memorare data citita
    MOV    AL,     0
KEYL: INC    AL            ; AL = AL + 1
    ROR    CX,     1        ; rotire CX dreapta cu 1, prin CF
    JC     KEYL            ; reia, daca B0 ≠ 0: (CY = 1)
; la iesire s-a gasit bit = 0; AX = cod tasta + 1
    DEC    AL              ; corectie AL = cod hexa tasta numerica
;
    RET
;-----
NMIERR:
; Rutina tratare NMI (eroare de OK convertizor)
;
; salvare registre in stiva
    PUSH    AX
    PUSH    BX
    PUSH    CX
    PUSH    DX
; ca la ERROK
    MOV     AX,     11000B
    OUT     PCMD, AX        ; comenzi
    MOV     AX,     0E111H
    CALL    AFISAJ         ; afisare mesaj eroare
;
; refacere registre din stiva
    POP     DX

```

```

    POP    CX
    POP    BX
    POP    AX
;
    RETI
;-----
SRTC0:
; Rutina tratare intrerupere de la RTC, timer 0. Este lansata la fiecare perioada
; de esantionare (1 ms)
;
; salvare registre in stiva
    PUSH   AX
    PUSH   BX
    PUSH   CX
    PUSH   DX
;
; citire pozitie curenta
    IN     AX,    POZITIA    ; citire date encoder
    SUB    AX,    POZCOR     ; corectie pozitie
    MOV    POZCRT,    AX     ; memorare pozitia curenta
; citire viteza curenta de la tahogenerator
    OUT    PADSTART,    AX    ; start conversie A/D
WDA: IN     AX,    PADDATA    ; citire stare si data de la A/D
    MOV    BX,    AX         ; salvare in BX
    AND    AX,    8000H      ; izolare bit "Data Ready" (B15)
    JNZ    WDA              ; reia, daca conversie in curs de desfasurare
    MOV    VITCRT,    BX     ; data valida, se memoreaza (conversia e terminata)
;
; calcul variatie pozitie
    MOV    AX,    POZCRT     ; AX =  $\Theta$ 
    SUB    AX,    POZIM1     ; AX =  $\Delta\Theta$ 
    MOV    DPOZ,    AX       ; memorare  $\Delta\Theta$ 
;
; calcul variatie viteza
    MOV    BX,    VITCRT     ; BX =  $\Omega$ 
    SUB    BX,    VITIM1     ; BX =  $\Delta\Omega$ 
    MOV    DVIT,    BX       ; memorare  $\Delta\Omega$ 
;
; calcul eroare de pozitie
    MOV    CX,    POZREF     ; CX =  $\Theta^*$ 
    SUB    CX,    POZCRT     ; CX =  $\varepsilon\Theta$ 
    MOV    PERROR,    CX     ; memorare  $\varepsilon\Theta$ 
;
; calcul comanda ui
; ui = ui-1 + F1* $\varepsilon$  + F2* $\Delta\Omega$  + F3* $\Delta\Theta$ 
; F1, F2 si F3 sunt reprezentate pe 8 biti fiecare
;
; calcul F1* $\varepsilon$ 
    MOV    AL,    F1         ; AL = F1
    IMUL    PERROR          ; AX = F1 *  $\varepsilon$ 
    MOV    CX,    AX         ; memorare in CX
;
; calcul F2* $\Delta\Omega$ 
    MOV    AL,    F2         ; AL = F2

```

```

IMUL DVIT      ; AX = F2*ΔΩ
ADD  CX,  AX   ; CX = F1*ε + F2*ΔΩ
;
; calcul F3*ΔΘ
MOV  AL,  F3    ; AL = F3
IMUL DPOZ      ; AX = F3*ΔΘ
ADD  CX,  AX    ; CX = F1*ε + F2*ΔΩ + F3*ΔΘ
;
MOV  AX,  UIM1  ; AX = ui-1
ADD  AX,  CX    ; AX = ui = ui-1 + F1*ε + F2*ΔΩ + F3*ΔΘ
;
; aici, eventual saturare comanda ui (neimplementata)
; ...
;
; comanda spre convertizor (prin convertorul D/A)
MOV  UCRT, AX   ; memorare comanda ui
OUT  PDADATA,  AX
;
; setari pentru pasul urmator
MOV  AX,  POZCRT ; AX = ΘI
MOV  POZIM1,  AX ; ΘI-1 = AX
;
MOV  AX,  VITCRT ; AX = ΩI
MOV  VITIM1, AX ; ΩI-1 = AX
;
MOV  AX,  UCRT   ; AX = ui
MOV  UIM1,  AX   ; ui-1 = AX
;
; refacere registre din stiva
POP  DX
POP  CX
POP  BX
;
; setare sfarsit de intrerupere, EOI catre 8259A
MOV  AL,  20H
OUT  PICOCW23, AL ; EOI catre 8259A
;
POP  AX
;
RETI
;-----
;
; Tabelul de coduri pentru cele 16 cifre hexazecimale (coduri segmente de afisaj aprinse)
;
TABELH:
DB   3FH, 03H, 76H, 67H ; "0", "1", "2", "3"
DB   4BH, 6DH, 7DH, 47H ; "4", "5", "6", "7"
DB   7FH, 6FH, 5FH, 79H ; "8", "9", "A", "b"
DB   3CH, 73H, 7CH, 5CH ; "C", "d", "E", "F"
;
; Exemplu: codul cifrei "5"
;
;      c
;      ==
;      //

```

```

;      d // g
;      ====
;      // a
;      f //
;      ====
;      DP g f e d c b a
;      0 1 1 0 1 1 0 1 ≡ 6DH
;
;-----
;      Coeficientii de comanda pentru controlul dupa stare.
;      Fie valorile reprezentate pe 8 biti (obtinute prin proiectarea regulatorului
;      discret cu reactie dupa stare si scalarile necesare) :
;
F1:    DB    1
F2:    DB    5
F3:    DB   -3
;
;-----
;      END                      ; directiva asamblor pentru incheierea fisierului sursa
;-----

```

6.4.4 Concluzii

În concluzie, aplicatia propusa reprezinta un exemplu tratat complet, incepand cu enuntarea temei de proiectare, o etapa de analiza a structurii, a caracteristicilor si a parametrilor elementelor componente ale sistemului ce va implementa cerintele impuse, urmata de modelarea sistemului controlat, prezentarea principiala a problematicii proiectarii regulatorului discret, proiectarea structurii hardware si apoi implementarea structurii software aferente.

Gradul relativ înalt de complexitate al aplicatiei si performantele ridicate cerute sistemului numeric au impus, practic, baleirea în aceasta abordare a întregii problematice cu care se confrunta proiectantul unei aplicatii industriale a sistemelor microprocesor. Evident, prezentarea facuta nu atinge toate problemele de detaliu tehnic, cum ar fi analiza semnalelor în schema digitala (temporizari, *fan-out* etc.), probleme de realizare fizica a cablajelor si a circuitelor electronice de acest tip (proiectarea cablajelor, elaborarea schemelor, tehnologii de implementare a circuitelor integrate etc.), de punere în functiune si metode specifice de testare, hardware si software, a acestor sisteme. Scopul urmarit a fost acela de a da o cat mai corecta si completa evaluare a gamei de probleme care apar, a metodelor de baza legate de proiectarea acestor sisteme.

Consideram ca prin aceasta abordare se poate ajunge la un grad de intelegere a principiilor de functionare si utilizare ale acestor sisteme. În consecinta, estimand ca majoritatea cititorilor acestor randuri sunt, cel mai probabil, potentiali utilizatori ai unor asemenea sisteme, care vor trebui sa implementeze sau sa modifice aplicatii industriale ale unor configuratii hardware specificate sau selectabile într-o gama relativ precizata de structuri posibile, deci sa elaboreze programele pentru aceste sisteme, scopul urmarit a fost de a sublinia elementele esentiale pe baza carora se poate aborda o asemenea problematica. Doar intelegand principiile de func-

tionare si organizare a acestor sisteme, se vor putea elabora programe eficiente, bine proiectate, flexibile, deschise spre modificari si/sau dezvoltari ulterioare.

Prezentarea facuta poate fi, totodata, si un solid punct de pornire pentru studierea, intelegerea si utilizarea practica a unor alte tipuri de microprocesoare utilizate din ce in ce mai mult in aplicatiile industriale. Reamintim aici doua categorii foarte importante: microcontrolerele industriale si procesoarele de semnal, DSP. Utilizand conceptele de baza si principiile de proiectare hardware si software introduse in prezenta lucrare, se va putea trece relativ usor la studiul si utilizarea acestor echipamente.

Nu putem incheia fara a sublinia un element fundamental legat de problema invatarii elementelor prezentate in aceasta lucrare: *SINGURA cale pentru a putea intelege cu adevarat si a invata ce sunt, cum functioneaza, cum se proiecteaza si cum se programeaza sistemele cu microprocesoare este de a LUCRA EFECTIV cu asemenea sisteme*, a le testa, a scrie, a depana si a dezvolta programe de aplicatii. Doar astfel se poate ajunge efectiv de la stadiul unei intelegeri reale, la experienta si indemanarea tehnica necesare pentru a deveni un utilizator eficient al acestor echipamente, ce pot si trebuie sa devina pentru specialistul modern “rigla de calcul”, instrumentul indispensabil in rezolvarea problemelor ingineresti celor mai complexe.